The Grid Analysis and Display System

GrADS

V1.5.1.12

Brian Doty

doty@cola.iges.org

10 September, 1995

Manual reformatted and updated by:

Tom Holt,
Climatic Research Unit,
University of East Anglia, Norwich, UK.
t.holt@uea.ac.uk

and

Mike Fiorino
Program for Climate Model Diagnosis and Intercomparison
Lawrence Livermore National Laboratory L-264
Livermore, CA 94551
fiorino@typhoon.llnl.gov

Table of Contents

Abstract

The Grid Analysis and Display System (GrADS) is an interactive desktop tool that is currently in use worldwide for the analysis and display of earth science data. GrADS is implemented on all commonly available UNIX workstations and DOS based PCs, and is freely distributed over the Internet. GrADS provides an integrated environment for access, manipulation, and display of earth science data.

GrADS implements a 4-Dimensional data model, where the dimensions are usually latitude, longitude, level, and time. Each data set is located within this 4-Dimensional space by use of a data description file. Both gridded and station data may be described. Gridded data may be non-linearly spaced; Gaussian grids and variable resolution ocean model grids are directly supported. The internal data representation in a file may be either binary or GRIB.

Since each data set is located within the 4-D data space, intercomparison of disparate data sets is greatly facilitated. Operations may be performed between data on different grids, or between gridded and observational data. Data from different data sets may be graphically overlaid, with correct spatial and time registration.

The user accesses data from the perspective of the 4-D data model. A dimension environment is described by the user as a desired subset of the 4-D space. Data is accessed, manipulated, and displayed within this subset.

Operations may be performed on the data directly, and interactively, by entering FORTRAN-like expressions at the command line. A rich set of built-in functions are provided. In addition, users may add their own functions as external routines written in any programming language. The expression syntax allows complex operations that range over very large amounts of data to be performed with simple expressions.

Once the data have been accessed and manipulated, they may be displayed using a variety of graphical output techniques, including line, bar, and scatter plots, as well as contour, shaded contour, streamline, wind vector, grid box, shaded grid box, and station model plots. Graphics may also be output in PostScript format for printing on monochrome or color PostScript printers. The user has wide control over all aspects of graphics output, or may choose to use the geophysically intuitive defaults.

A programmable interface is provided in the form of an interpreted scripting language. A script may display widgets as well as graphics, and take actions based on user point-and-clicks. Quite sophisticated data graphical interfaces can, and have, been built. The scripting language can also be used to automate complex multi-step calculations or displays. GrADS can be run in a batch mode, and the scripting language facilitates using GrADS to do long overnight batch jobs.

Development plans for 1995 include a Microsoft Windows implementation, support for geographically registered image data, and development of an interface to BUFR data sets. In addition, we plan to implement a number of user requested features, such as arbitrary vertical cross sections, an interface to the NetCDF data storage package, and an enhanced point-and-click help facility.

Suggestions

Please forward any suggestions you have for improving GrADS to me **doty@cola.iges.org**. I am always interested in hearing what you like and don't like about GrADS, and am always looking for ways to improve it.

We also recommend that you joint the gradsusr listserver described in Appendix F. This forum is also monitored by the GrADS development community and is another channel form making suggestions.

How to use this Manual

This manual is divided into three sections:

- Introductory Guide
- Reference Section
- Appendices

Introductory Guide

The Introductory Guide provides a conceptual framework within which the more detailed information contained in subsequent sections can be absorbed as needed. Thus, this section contains most of the information needed to run GrADS at a rudimentary level. However, it is not designed to stand alone and users will certainly need to refer to some of the material in the Reference Section almost immediately.

Reference Section

The first two chapters in this section contain detailed descriptions of all the options available for graphical display of data, followed by a reference to GrADS functions. Both these chapters are organised by functional category. This enables users to decide what they want to do and then quickly refer to all the options available to them. The remaining chapters in this section provide information for more advanced use of GrADS and a deeper understanding of the processes described in the Introductory Guide.

Appendices

The appendices contain information considered not immediately relevant to earlier chapters. This includes some platform-specific information and ancillary material which could reduce the time taken to become familiar with GrADS facilities. All users are advised to browse the appendices before using GrADS.

Using the Manual

Instead of an index, this manual uses a detailed **Table of Contents** to help find information. Users are recommended to refer to this if they are unsure how to do something. We have attempted to organise the manual in a functional manner to reduce the time spent reading irrelevant information. It is suggested that new users should read the first three chapters of the Introductory Guide, and then experiment with the sample data file described in Chapter 3 **Tutorial** before using GrADS on your own data. You should then examine the sample data description files listed in Chapter 4 **Using GrADS Data Files**. Following the instructions in this chapter, it should be a fairly simple matter to construct a test data description file for a small sample set of data. This can then be experimented on using GrADS by referencing the appropriate chapters and gradually expanding your awareness of the capabilities of GrADS.

Introductory Guide

1.0 Starting and Quitting GrADS

GrADS is started by entering the command: **grads**

Before initialising the graphics output environment, GrADS will prompt for landscape or portrait mode. Landscape is 11 x 8.5 inches (usually what you want). Portrait is 8.5 x 11 inches, primarily used for producing vertically oriented hardcopy output. The actual size of the window will not, of course, be 11 x 8.5 inches (or 8.5 x 11 inches), but instead will be whatever size you chose by using your workstation's window manager. But GrADS will treat the window as if it were one of the above sizes, so it is best to size the window with approximately the proper aspect ratio. This can be done using the window manager or from GrADS using the command:

set xsize x y

which resizes the window to **x,y** pixels.

After answering this prompt, a separate graphics output window will be opened (but not on PCs). You may move or resize this window at any time.

You will enter GrADS commands in the text window from where you started GrADS. Graphics output will appear in the graphics window in response to the commands you enter. You will thus need to make the text window the "active" window; the window that receives keyboard input.

Help

Typing **help** at the GrADS command prompt gives a summary list of operations essential to do anything in GrADS. This is intended to jog memory rather than provide an exhaustive help facility. If the GrADS manual is not available, you can obtain info on most command parameters by typing the command on its own. Alternatively, we are setting up comprehensive documentation intended to be used as a *local* Web facility.

Diagnostics at startup

When you start GrADS you get platform specific diagnostics, for example:

GX Package Initialization: Size = 11 8.5
!!!! 32-bit BIG ENDIAN machine version
ga>

The **!!!!** line tells you that this version is **32-bit** (i.e., data are 32-bit) and it was compiled for a **big endian** machine (the Sun in this case). On the Cray you get...

!!!! 64-BIT MACHINE VERSION (CRAYS)

Startup options

You may specify the following options as arguments to the **'grads'** command when GrADS is started:

b Run grads in batch mode. No graphics output window is opened.
l Run grads in landscape mode. The Portrait vs. Landscape question is not asked.
p Run grads in portrait mode.
c Execute the supplied command as the 1st GrADS command after GrADS is started.

An example:

grads -c "run profile.gs"

These options may be used in combinations. For example:

grads -blc "run batch.gs"

Would run grads in batch mode, using landscape orientation (thus no questions are asked at startup); and execute the command:

"**batch.gs**" upon startup.

Leaving GrADS

To leave GrADS, enter the command:

quit

2.0 Basic Concept of Operation

When you have successfully installed and started GrADS, you'll be confronted with two windows[1] -- a terminal window with a prompt, much like the infamous C:> in MS-DOS, and a resizable window (black background by default) where graphics are displayed.

GrADS commands are entered in the terminal window and the response from GrADS is either graphics in the graphics window or text in the terminal window. The three fundamental GrADS commands:

- **open** open or make available to GrADS a data file with either gridded or station data
- **d** display a GrADS "expression" (e.g., a slice of data)
- **set** manipulate the "what" "where" and "how" of data display

The GrADS "expression," or what you want to look at, can be as simple as a variable in the data file that was **open**ed, e.g., '**d** slp' or an arithmetic or GrADS function operation on the data, e.g., '**d** slp/100' or '**d** *mag*(u,v)' where *mag* is a GrADS intrinsic function.

The "where" of data display is called the "dimension environment" and defines which part, chunk or "hyperslab" of the 4-D geophysical space (lon,lat,level,time) is displayed. The dimension environment is manipulated through the **set** command and is controlled in either *grid* coordinates (x,y,z,t or indices) or *world* coordinates (lon, lat,lev, time).

The "what" and "how" of display is controlled by **set** commands and includes both graphics methods (e.g., contours, streamlines) and data (e.g., **d** to a file).

GrADS graphics can be written to a file (i.e., **enable print** filename and **print**) and then converted to postscript for printing and/or conversion to other image formats.

In addition, GrADS includes graphic primitives (e.g., lines and circles) and basic labelling through the **draw** command.

The **q** or **query** command is used to get information from GrADS such as which files are opened and even statistics.

[1] With the exception of the MS-DOS version which only has one window (the only non-X windows version)

3.0 Tutorial

A tutorial and sample data is available from the distribution sites (ftp://grads.iges.org/example.tar and ftp://sprite.llnl.gov/pub/fiorino/grads/example). Portions of the sample.txt are included below as another starting point for new GrADS users:

The following sample session will give you a feeling for how to use the basic capabilities of GrADS. You will need the data file 'model.dat' on your system. This sample session takes about 30 minutes to run through.

This data file is described by the data descriptor file 'model.ctl'. You may want to look at this file before continuing. The data descriptor file describes the actual data file, which in the case contains 5 days of global grids that are 72 x 46 elements in size.

To start up GrADS, enter:

grads

If grads is not in your current directory, or if it is not in your PATH somewhere, you may need to enter the full pathname, ie:

/usr/homes/smith/grads/grads

GrADS will prompt you with a landscape vs. portrait question; just press enter. At this point a graphics output window should open on your console. You may wish to move or resize this window. Keep in mind that you will be entering GrADS commands from the window where you first started GrADS -- this window will need to be made the 'active' window and you will not want to entirely cover that window with the graphics output window.

In the text window (where you started grads from), you should now see a prompt: ga> You will enter GrADS commands at this prompt and see the results displayed in the graphics output window.

The first command you will enter is:

open model.ctl

You may want to see what is in this file, so enter:

query file

One of the available variable is called **ps**, for surface pressure. We can display this variable by entering:

d ps

d is short for display. You will note that by default, GrADS will display an X, Y plot at the first time and at the lowest level in the data set.

Now you will enter commands to alter the 'dimension environment'. The display command (and implicitly, the access, operation, and output of the data) will do things with respect to the current dimension environment. You control the dimension environment by entering **set** commands:

clear	clear the display
set lon -90	set longitude fixed
set lat 40	set latitude fixed
set lev 500	set level fixed
set t 1	set time fixed
d z	display a variable

In the above sequence of commands, we have set all four GrADS dimensions to a single value. When we set a dimension to a single value, we say that dimension is fixed. Since all the dimensions are fixed, when we display a variable we get a single value, in this case the value at the location 90W, 40N, 500mb, and the 1st time in the data set.

If we now enter:

set lon -180 0	**X** is now a varying dimension
d z	

We have set the X dimension, or longitude, to vary. We have done this by entering two values on the set command. We now have one varying dimension (the other dimensions are still fixed), and when we display a variable we get a line graph, in this case a graph of 500mb Heights at 40N.

Now enter:

clear
set lat 0 90
d z

We now have two varying dimensions, so by default we get a contour plot. If we have 3 varying dimensions:

c
set t 1 5
d z

we get an animation sequence, in this case through time.

Now enter:

clear
set lon -90
set lat -90 90
set lev 1000 100
set t 1
d t
d u

In this case we have set the Y (latitude) and Z (level) dimensions to vary, so we get a vertical cross section. We have also displayed two variables, which simply overlay each other. You may display as many items as you desire overlaid before you enter the clear command.

Another example, in this case with X and T varying (Hovmoller plot):

c
set lon -180 0
set lat 40
set lev 500
set t 1 5
d z

Now that you know how to select the portion of the data set to view, we will move on to the topic of operations on the data. First, set the dimension environment to an Z, Y varying one:

clear
set lon -180 0
set lat 0 90
set lev 500
set t 1

Now lets say that we want to see the temperature in Fahrenheit instead of Kelvin. We can do the conversion by entering:

display (t-273.16)*9/5+32

Any expression may be entered that involves the standard operators of +, -, *, and /, and which involves operands which may be constants, variables, or functions. An example involving functions:

clear
d sqrt(u*u+v*v)

to calculate the magnitude of the wind. A function is provided to do this calculation directly:

d mag(u,v)

Another built in function is the averaging function:

clear
d ave(z,t=1,t=5)

In this case we calculate the 5 day mean. We can also remove the mean from the current field:

d z - ave(z,t=1,t=5)

We can also take means over longitude to remove the zonal mean:

clear
d z-ave(z,x=1,x=72)
d z

We can also perform time differencing:

clear
d z(t=2)-z(t=1)

This computes the change between the two fields over 1 day. We could have also done this calculation using an offset from the current time:

d z(t+1) - z

The complete specification of a variable name is:

name.file(dim +|-|= value, ...)

If we had two files open, perhaps one with model output, the other with analyses, we could take the difference between the two fields by entering: display z.2 - z.1

Another built in function calculates horizontal relative vorticity via finite differencing:

clear
d hcurl(u,v)

Yet another function takes a mass weighted vertical integral:

clear
d vint(ps,q,275)

Here we have calculated precipitable water in mm.

Now we will move on to the topic of controlling the graphics output. So far, we have allowed GrADS to chose a default contour interval. We can override this by:

clear
set cint 30
d z

We can also control the contour color by:

clear
set ccolor 3
d z

We can select alternate ways of displaying the data:

clear
set gxout shaded
d hcurl(u,v)

This is not very smooth; we can apply a cubic smoother by entering:

clear
set csmooth on
d hcurl(u,v)

We can overlay different graphics types:

set gxout contour
set ccolor 0
set cint 30
d z

and we can annotate:

draw title 500mb Heights and Vorticity

We can view wind vectors:

clear
set gxout vector
d u;v

Here we are displaying two expressions, the first for the U component of the vector; the 2nd the V component of the vector. We can also colorize the vectors by specifying a 3rd field:

d u;v;q

or maybe:

d u;v;hcurl(u,v)

You may display pseudo vectors by displaying any field you want:

clear
d mag(u,v) ; q*10000

Here the U component is the wind speed; the V component is moisture.

We can also view streamlines (and colorize them):

clear
set gxout stream
d u;v;hcurl(u,v)

Or we can display actual grid point values:

clear
set gxout grid
d u

We may wish to alter the map background:

clear	
set lon -110 -70	
set lat 30 45	
set mpdset nam	
set digsiz 0.2	Digit size
set dignum 2	# of digits after decimal place
d u	

To alter the projection:

set lon -140 -40	
set lat 15 80	
set mpvals -120 -75 25 65	Map projection constants
set mproj nps	North Polar Stereographic
set gxout contour	
set cint 30	
d z	

In this case, we have told grads to access and operate on data from longitude 140W to 40W, and latitude 15N to 80N. But we have told it to display a polar stereographic plot that contains the region bounded by 120W to 75W and 25N to 65N. The extra plotting area is clipped by the map projection routine.

This concludes the sample session. At this point, you may wish to examine the data set further, or you may want to go through the GrADS documentation and try out the other options described there.

4.0 Using GrADS Data Files

GrADS supports two basic data types:

- gridded data — data on a **grid**
- station data — **station** or point observations.

The data and meta data (or information about the data) are kept in separate files. The meta data file contains a complete description of the data and the name of the file containing it, the data file is purely data with no space or time identifiers. The file which you open in GrADS is the **data descriptor** file (the meta data) or **.ctl** file. The .ctl is constructed to describe various data types and structures (e.g., binary and GRIB).

You will need to open at least one data-descriptor file before you can enter other GrADS commands.

open filename

You can open more than one data-descriptor file. Each file is numbered in the order that you open it in. Initially, the "default file" is file 1, or the first file you open. The importance of the default file will be discussed later.

Default file extension

"**.ctl**" is the *de facto* standard extension for GrADS data descriptor files. Provided you adhere to this standard there is no need to type the extension ".ctl" when issuing the **open** command. For example. typing:

open jandata.1966 has the same effect as **open jandata.1966.ctl**

Introduction to GrADS Data Sets

The raw data are on disk in either binary direct access or sequential unformatted form (IEEE floats and ints) or GRIB.

The data are described by the **data descriptor file**, which contains:

- Name of the binary data set.
- Mapping between grid coordinates and world coordinates.
- Number of variables, abbreviations for variables.

The data-descriptor file is free format, where each field is blank delimited. It can be created easily with a text editor. The data descriptor file assigns a one to twelve character abbreviation for each variable in the file. These abbreviations are used in GrADS expressions.

The use of data descriptor files is now discussed for gridded and station data. This material uses simple examples which should be enough to enable users to explore the capabilities of GrADS. More advanced features of .ctl files are described in **Chapter 22** in the **Reference Section**.

Gridded Data Sets

The GrADS gridded may contain any number of variables at specified **longitude**, **latitude**, **vertical levels**, and **time intervals**. Latitudes can vary from *north to south* or from *south to north* (the default), and levels can vary from *top to bottom* or from *bottom to top*.

GrADS views this data set as a giant "5-D" array—with **X** (longitude or lon) varying the fastest, then **Y** (latitude or lat), then **Z** (vertical level or lev), then the **variable type**, then **T** (time).

It is easier for us to think of the data set in terms of a sequence of horizontal grids, where longitude and latitude vary. Each horizontal grid represents a particular variable at a particular height and time. Each horizontal grid is the same size in any particular GrADS data set (if you have grids of different sizes, you must create separate data sets).

These grids are written to the data set in the following order: starting with a particular variable, grids for each vertical level (at a particular time) are written out in ascending order. Then the grids for the next variable are written out. When all the grids at a particular time have been written, grids for the next time are written.

The format of this data set is thus exactly the same as the COLA Pressure History format, except: there are no date/time records and, by default, latitude varies from south to north (not north to south as in the pressure history data).

Each binary gridded data set is described by a separate data descriptor file, essentially a table of contents for the binary data set. Following is an example of a simple data descriptor file:

```
DSET       ua.dat
TITLE      Upper Air Data
UNDEF      -9.99E33
OPTIONS BYTESWAPPED
XDEF       80 LINEAR  -140.0  1.0
YDEF       50 LINEAR   20.0   1.0
ZDEF       10 LEVELS  1000 850 700 500 400 300 250 200 150 100
TDEF       4 LINEAR 0Z10apr1991  12hr
VARS       6
   slp     0    0  sea level pressure
   z       10    0  heights
   t       10    0  temps
   td      6    0  dewpoints
   u       10    0  u winds
   v       10    0  v winds
ENDVARS
```

The data descriptor file is **'free format'**, ie each entry is blank delimited and may appear in any column. **Comment** records start with an asterisk (**'*'**) in column 1. Comments may not appear in the list of variable records (between the **vars** and **endvars** records). Records may not be more than 255 characters long.

In this example, the binary data set is named **ua.dat**, the undefined, or missing, data value is **-9.99e33**, there are **80** grid points in the X direction, **50** in the Y direction, **10** levels, **4** times, and **6** variables. The variables **z, t, u**, and **v** have **10** levels, the variable **td** has **6** levels, and the variable **slp** has **one** level (see below for a more specific description of each entry).

Think in terms of the X and Y data points at one level for one variable at one time being a horizontal grid. This grid is exactly in the same storage order as a FORTRAN array, in this case an array

DIMENSION A(80,50). The first dimension always varies from west to east, the second from south to north (by default).

In the above example the horizontal grids would be written in the following order:

```
Time 1, Level    ?, Variable slp
Time 1, Level 1000, Variable z
Time 1, Level  850, Variable z
    then levels 700, 500, 400, 300, 250, 200, then
Time 1, Level  150, Variable z
Time 1, Level  100, Variable z
Time 1, Level 1000, Variable t
Time 1, Level  850, Variable t
    then levels 700, 500, 400, 300, 250, 200, then
Time 1, Level  150, Variable t
Time 1, Level  100, Variable t
Time 1, Level 1000, Variable td
Time 1, Level  850, Variable td
Time 1, Level  700, Variable td
Time 1, Level  500, Variable td
Time 1, Level  400, Variable td
Time 1, Level  300, Variable td
Time 1, Level 1000, Variable u
Time 1, Level  850, Variable u
    then levels 700, 500, 400, 300, 250, 200, then
Time 1, Level  150, Variable u
Time 1, Level  100, Variable u
Time 1, Level 1000, Variable v
Time 1, Level  850, Variable v
    then levels 700, 500, 400, 300, 250, 200, then
Time 1, Level  150, Variable v
Time 1, Level  100, Variable v
Time 2, Level    ?, Variable slp
Time 2, Level 1000, Variable z
Time 2, Level  850, Variable z
Time 2, Level  700, Variable z
Time 2, Level  500, Variable z
Time 2, Level  400, Variable z
        .
        etc
```

A description of each record in the example GrADS gridded data descriptor file follows:

DSET data-set-name

This entry specifies the name of the binary data set. It may be entered in mixed case.

If the binary data set is in the same directory as the data descriptor file, you may enter the filename in the data descriptor file without a full path name by prefixing it with a ^ character. For example, if the data descriptor file is:

/data/wx/grads/sa.ctl

and the binary data file is:

/data/wx/grads/sa.dat

you could use the following file name in the data descriptor file:

DSET ^sa.dat

instead of:

DSET /data/wx/grads/sa.dat

As long as you keep the two files together, you may move them to any directory without changing the entries in the data descriptor file.

TITLE string

A brief description of the contents of the data set. This will be displayed during a **query** command, so it is helpful to put meaningful information here.

UNDEF value

The undefined, or missing, data value. GrADS operations and graphics routines will ignore data with this value from this data set. *This is a required parameter even if there are no undefined data*.

OPTIONS BYTESWAPPED

Indicates the binary data file is in reverse byte order from the normal byte order of the machine. This would happen if you sent a file in binary format from, for example, a Sun to a PC. Putting this keyword in the descriptor file tells GrADS to swap the byte order as the data is being read.

XDEF number <LINEAR start increment> or <LEVELS value-list>

Defines the mapping between grid values and longitude. Specifically:

number -- the number of grid values in the X direction, specified as an integer number. Must be **>= 1**.

LINEAR or LEVELS **-** Indicates the grid mapping type.

For **LINEAR**:

- **start** -- the starting longitude, or the longitude for X=1. Specified as a floating point value, where negative indicates degrees west.
- **increment** -- the spacing between grid value in the X direction. It is assumed that the X dimension values go from west to east. Specified as a positive floating value.

For **LEVELS**:

- **value-list** -- List of 'number' values representing the longitude of each X dimension. May start and continue on the next record in the descriptor file (records may not be > 255 characters). There must be at least 2 levels (otherwise use **LINEAR** mapping).

YDEF number mapping start <increment>

<LEVELS value-list>

Defines the mapping between grid values and latitude. Specifically:

number -- the number of grid values in the X direction, specified as an integer number.
mapping -- mapping type, specified as a keyword.

Valid are:

- **LINEAR** -- Linear mapping
- **GAUSR15** -- Gaussian R15 latitudes

GAUSR20 -- Gaussian R20 latitudes
GAUSR30 -- Gaussian R30 latitudes
GAUSR40 -- Gaussian R40 latitudes

Examples of specifying **GAUSRxx** mapping:

YDEF 20 GAUSR40 15

Indicates that there are 20 Y dimension values which start at Gaussian Latitude 15 (64.10 south) on the Gaussian R40 grid. Thus the 20 values would correspond to Latitudes:

64.10, -62.34, -60.58, -58.83, -57.07, -55.32, -53.56,
51.80, -50.05, -48.29, -46.54, -44.78, -43.02, -41.27,
39.51, -37.76, -36.00, -34.24, -32.49, -30.73

YDEF 102 GAUSR40 1

The entire gaussian grid is present, starting at the southernmost latitude (-88.66).

start -- For **LINEAR** mapping, the starting latitude, ie the latitude for Y = 1, and is specified as a floating point value, with negative indicating degrees south. For **GAUSRxx** mapping, the start value indicates the first gaussian grid number, where 1 would be the southernmost gaussian grid latitude.

increment -- the spacing between grid values in the Y direction. It is assumed that the Y dimension values go from south to north. Specified as a positive floating point value. Used only for **LINEAR** mapping.

For LEVELS:

value-list -- List of 'number' values representing the latitude of each X dimension. May start and continue on the next record in the descriptor file (records may not be > 80 characters). There must be at least 2 levels (otherwise use **LINEAR** mapping).

ZDEF number mapping <start increment>

<value-list>

Defines the mapping between grid values and pressure level. Specifically:

number -- the number of grid values in the X direction, specified as an integer number.
mapping -- mapping type, specified as a keyword.
Valid are:
LINEAR -- Linear mapping
LEVELS -- Arbitrary pressure levels
start -- when mapping is **LINEAR**, this is the starting value, or the value when **Z=1**.
increment -- when mapping is **LINEAR**, the increment in the Z direction, or from lower to higher. This may be a negative value, for example:

ZDEF 10 LINEAR 1000 -100

indicating that the data is for levels 1000, 900, 800, 700, etc.

value-list -- when the mapping is **LEVELS**, the specific levels are simply listed in ascending order. If there is only one level, use **LINEAR**, since **LEVELS** implies at least two levels.

TDEF number LINEAR start-time increment

Defines the mapping between grid values and time. Specifically:

number -- the number of times in the data set. Specified as an integer number.

start-time -- The starting date/time value, specified in GrADS absolute date/time format. This is the value when T=1. The date/time format is:

hh:mmZddmmmyyyy where:

hh = hour (two digit integer)
mm = minutes (two digit integer)
dd = day (one or two digit integer)
mmm = month (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec)
yyyy = year (two or four digit integer. two digits implies a year between 1950 and 2049).

If not specified, **hh** defaults to **00**, **mm** defaults to **00**, and **dd** defaults to **1**. The month and year must be specified. No intervening blanks are allowed in a GrADS absolute date/time.

Examples:

12Z1JAN1990
14:20Z22JAN1987
JUN1960

increment -- time increment. Specified in GrADS time increment format:

vvkk where:

vv = an integer number, 1 or 2 digits
kk = an increment keyword,
mn = minutes
hr = hours
dy = days
mo = months
yr = year

Examples:

20mn -- increment is 20 minutes
1mo -- increment is 1 month
2dy -- increment is 2 days

Further examples of a TDEF statement:

TDEF 24 LINEAR 00Z01JUN1987 1HR

The data set has **24** times, starting at **00Z** on **1 Jun**, **1987**, with an increment of **1 hour**.

TDEF 30 LINEAR 2JUN1988 1DY

The data set has **30** times, starting at **00Z** on **2 Jun**, **1988**, with an increment of **1 day**.**]**

VARS number

Indicates the start of the records describing the variables in the data set.

number -- the number of variable records

variable records (slp ... v)

There are six variable records in this example, each with the following format:

abrev levs units description

abrev -- a 1 to 12 character abbreviation for this variable. This abbreviation must start with an alphabetic character (a-z) and be composed of alphabetic characters and numbers. This abbreviation will be the "name" the variable is accessed by from within GrADS.

levs -- an integer value specifying the number of levels this variable has in the data set. It may not exceed the number of levels in the ZDEF statement. A levs value of 0 indicates this variable has one "level" that does not correspond to a vertical level. An example would be a surface variable.

units - Used for GRIB data and special data format/structures. Put a value of 99 here.

description - A text description of the variable, max 40 characters.

ENDVARS

After the last variable record comes the **ENDVARS** statement. This ends the GrADS data descriptor file. ***Blank lines after the ENDVARS statement may cause GrADS open to fail!***

The options record in the Data Descriptor File

The **options** record in the data descriptor file allows you to control various aspects of the way GrADS interprets your raw data file. It obsoletes the old "**format**" record and has the form:

options <keywords>

Some keywords are:

options <yrev> <zrev> <sequential> <byteswapped> <template> <big_endian> <little_endian> <cray_32bit_ieee>

where:

sequential specifies that the file was written in sequential unformatted I/O, where each record is an X/Y varying grid. Note that if you have only one X and one Y dimension in your file, each record in the file will be one element long (it may not be a good idea to write the file this way).

yrev specifies that the Y, or latitude, dimension has been written in the reverse order from what GrADS has in the past assumed. An important thing to remember is that GrADS still presents the view that the data goes from south to north. The YDEF statement does not change; it still describes the transformation from a grid space going from south to north. The reversal of the Y axis is done as the data is being read from the data file.

zrev indicates the data has been written into the file from top to bottom, rather than from bottom to top as GrADS has in the past assumed. The same considerations as **yrev** apply.

template file name templates are in use (see the section on **Multiple File Time Series** in Chapter 22)

byteswapped byte ordering of data is reversed endian (see the next two options and the above example .ctl file)

The best way to ensure independence of hardware for gridded data files is to specify the source platform of the data. This allows the data to worked on both types of hardware without having to worry about byte ordering. The following two **option** parameters indicate the actual **byte ordering** of the data. If the data are already in the correct order, no conversion is performed. These options facilitate moving data files and descriptor files between machines.

big_endian 32-bit IEEE floats created on a **big_endian** platform (e.g., cray, sun, sgi and hp).

little_endian 32-bit IEEE floats created on a **little_endian** platform (e.g., iX86, and dec)

Station Data Sets

Station data sets are written to a binary file one report at a time. The only ordering required is that the station reports be grouped within the file into some time interval. For example, the time interval for upper air observations might be 12 hours. Please refer to **Chapter 16** for more general information about GrADS facilities for analysing and displaying station data.

Variables within each report are split into two groupings. Each variable is either a surface variable, thus can be reported at most once per report, or it is a level dependent variable, thus can be reported at a number of different levels within one report.

Byte-ordering control for station data files: You may now specify byteswapping (byteswapped, big_endian, or little_endian) for station data files. The stnmap utility, and GrADs, will perform the necessary conversion. Station map files must still be created on the machine where they are to be used.

The format of a station report in the binary station data file is:

- A header which provides information about the location of the station.
- Surface variables, if any
- Level dependent variables, if any

The header is described by the following C language data structure:

```
struct rpthdr {
  char id[8];          /* Character station id           */
  float lat;           /* Latitude of report             */
  float lon;           /* Longitude of report            */
  float t;             /* Time in relative grid units    */
  int  nlev;           /* Number of levels following     */
  int flag;            /* Level independent var set flag */
};
```

A detailed description of each header entry follows:

id - The station identifier. This is a 1 to 7 character identifier that should identify the station uniquely. It may be assigned arbitrarily; ie. the stations could be numbered in some arbitrary order.

lat - The Y dimension location of the station in world coordinates, typically latitude.

lon - The X dimension location of the station in world coordinates, typically longitude.

t - The time of this report, in relative grid units. This refers to the way the stations are grouped in time. For example, if you are working with surface airways reports, you would probably have a time grouping interval of one hour. If you wanted to treat the report times of each report as being exactly on the hour, you would set t to 0.0. If the report was for 12:15pm, and you were writing the time group for 12pm, you would set t to be 0.25. Thus, t would typically have the range of -0.5 to 0.5.

nlev - Number of data groups following the header. This is the count of the one surface group, if present, plus the number of level dependent groups. Is set to zero to mark the end of a time group in the file.

flag - If zero, there are no surface variables following the header. If one, then there are surface variables following the header.

Following the header, the data for this report is written. The first group of data would be all the surface variables if present. Whether or not the surface variable (if any) are present is determined by the flag in the header. If present, then all the surface variables must be written—missing variables should have the missing data value provided. Thus, each surface variable group will be the same size for each report in the file.

The surface variables are written out as floating point numbers. The ordering of the variables must be the same in each report, and is the ordering that will be given in the data descriptor file.

Following the surface variable group, any number of level dependent groups may be written. The number of total data groups is provided in the header. Each level dependent group must have all the level dependent variables present, even if they are filled with the missing data value. Thus, each level dependent group will be the same size for all levels and all reports in the file.

The level dependent group is written out as follows:

level -- floating point value giving the Z dimension value in world coordinates for this level.
variables -- The level dependent variables for this level.

After all the reports for one time grouping have been written, a special header (with no data groups) is written to indicate the end of the time group. The header has an nlev value of zero. The next time group may then start immediately after. A time group with no reports would still contain the time group terminator header record (ie, two terminators in a row).

GrADS station data files must be written as UNIX stream data sets without any imbedded record descriptor information. This is easily done from a C program. From a FORTRAN program, it usually requires a system-dependent option in the OPEN statement. For example, in DEC FORTRAN one can use the

RECORDTYPE='STREAM'

option to avoid having record descriptor information imbedded in the output file. Examples of C and FORTRAN programs to create station data sets are provided later in this document. *Because there are no standards for binary I/O in f77, it is strongly recommended station data conversion programs be written in C.*

Station Data Descriptor File

The format for the data descriptor file for station data is similar to the format for a gridded data set. An example of a station data descriptor file is:

```
dset  ^ua.reps
dtype  station
stnmap ^ua.map
undef  -999.0
title  Real Time Upper air obs
tdef  10 linear 12z18jan1992 12hr
vars        12
   slp      0  99  SLP
   ts  0  99  Temps
   ds       0  99  Dewpoints
   us       0  99  U Winds
   vs       0  99  V Winds
   z   1  99  Heights
   t   1  99  Temps
   d        1  99  Dewpoints
```

```
u          1  99  U Winds
v   1  99  V Winds
endvars
```

Note the differences between this descriptor file and a grid descriptor file:

DTYPE record -- specify a data type of: station.
STNMAP record -- gives the file name of the station mapping file. This file is created by the stnmap utility, which will be described later.
XDEF, YDEF, ZDEF records -- not specified.
TDEF record -- describes the time grouping interval and the number of time groups in the file.
VAR records -- surface variables must come first, and are given a zero for the number-of-levels field. Level dependent variables are listed after the surface variables, and are given a one in the number-of-levels field.

STNMAP Utility

Once the data set has been written, and the descriptor file created, you should then create the station map file by running the **stnmap** utility. This utility writes out hash table and/or link list information that allows GrADS to access the report data more efficiently. The utility will prompt for the name of the data descriptor file.

If you change the data file—perhaps by appending another time group—you will also have to change the descriptor file to reflect the changes—the new number of times for example -- and then rerun the **stnmap** utility.

Creating Data Files

This section describes how to create the raw data files for gridded and station data, with examples of appropriate data descriptor files.

Examples of Creating a Gridded Data Set

On a workstation, the binary GrADS data sets need to be created as a 'stream' data set, ie, it should not have the normal FORTRAN record descriptor words imbedded in it. This can be done from FORTRAN using direct access I/O:

```
      REAL  Z(72,46,16)
            .
            .
      OPEN (8,FILE='grads.dat',FORM='UNFORMATTED',
     &   ACCESS='DIRECT',RECL=72*46)
            .
            .
      IREC=1
      DO 10 I=1,18
        WRITE (8,REC=IREC) ((Z(J,K,I),J=1,72),K=1,46)
        IREC=IREC+1
10    CONTINUE
```

This example writes out 16 levels of one variable to a file in direct access format. We are really writing the data out sequentially, and using direct access to avoid having the record descriptor words written. There may be options in your compiler to do this more directly, or you may wish to write the data using a C program.

Another simple sample might be:

```
      REAL X(100)
      DO 10 I=1,100
        X(I)=I
 10   CONTINUE
      OPEN (8,FILE='samp.dat',FORM='UNFORMATTED',ACCESS='DIRECT',
     & RECL=100)
      WRITE (8,REC=1) X
      STOP
      END
```

The associated descriptor file:

```
DSET      samp.dat
TITLE     Sample Data Set
UNDEF     -9.99E33
XDEF      100 LINEAR 1 1
YDEF      1 LINEAR 1 1
ZDEF      1 LINEAR 1 1
TDEF      1 LINEAR 1JAN2000 1DY
VARS      1
  x  0  99  100 Data Points
ENDVARS
```

Once created, you can use this data set to experiment with GrADS data functions, such as:

display sin(x/50)

Examples of Creating Station Data Sets

Lets say you have a data set with monthly rainfall:

```
Year Month Stid        Lat    Lon     Rainfall

1980    1  QQQ         34.3   -85.5   123.3
1980    1  RRR         44.2   -84.5    87.1
1980    1  SSS         22.4   -83.5   412.8
1980    1  TTT         33.4   -82.5    23.3
1980    2  QQQ         34.3   -85.5   145.1
1980    2  RRR         44.2   -84.5   871.4
1980    2  SSS         22.4   -83.5   223.1
1980    2  TTT         33.4   -82.5    45.5
```

A FORTRAN program in DEC FORTRAN to write this data set in GrADS format might be:

```
      CHARACTER*8 STID
C
      OPEN (8,NAME='rain.ch')
      OPEN (10,NAME='rain.dat',FORM='UNFORMATTED',
    &  RECORDTYPE='STREAM')
C
      IFLAG = 0
C
C  Read and Write
C
10    READ (8,9000,END=90) IYEAR,IMONTH,STID,RLAT,RLON,RVAL
9000 FORMAT (I4,3X,I2,2X,A8,3F8.1)
      IF (IFLAG.EQ.0) THEN
        IFLAG = 1
        IYROLD = IYEAR
```

```
        IMNOLD = IMONTH
      ENDIF
C
C  If new time group, write time group terminator.
C  Assuming no empty time groups.
C
      IF (IYROLD.NE.IYEAR.OR.IMNOLD.NE.IMONTH) THEN
        NLEV = 0
        WRITE (10) STID,RLAT,RLON,TIM,NLEV,NFLAG
      ENDIF
      IYROLD = IYEAR
      IMNOLD = IMONTH
C
C  Write this report
C
      TIM = 0.0
      NLEV = 1
      NFLAG = 1
      WRITE (10) STID,RLAT,RLON,TIM,NLEV,NFLAG
      WRITE (10) RVAL
      GO TO 10
C
C  On end of file write last time group terminator.
C
 90   CONTINUE
      NLEV = 0
      WRITE (10) STID,RLAT,RLON,TIM,NLEV,NFLAG
      STOP
      END
```

For a different compiler, you would need the appropriate OPEN statement to write a stream data set, but this options is often times not available. *Support for sequential data is under consideration.*

An equivalent C program might be:

```
#include <stdio.h>
/* Structure that describes a report header in a stn file */
struct rpthdr {
  char id[8];           /* Character station id           */
  float lat;            /* Latitude of report             */
  float lon;            /* Longitude of report            */
  float t;              /* Time in relative grid units    */
  int  nlev;            /* Number of levels following     */
  int flag;             /* Level independent var set flag */
} hdr;

main () {
  FILE *ifile, *ofile;
  char rec[80];
  int flag,year,month,yrsav,mnsav,i;
  float val;

  /* Open files */
  ifile = fopen ("rain.ch","r");
  ofile = fopen ("rain.dat","wb");
  if (ifile==NULL || ofile==NULL) {
    printf ("Error opening files\n");
    return;
  }

  /* Read, write loop */
```

```
  flag = 1;
  while (fgets(rec,79,ifile)!=NULL) {
    /* Format conversion */
    sscanf (rec,"%i %i ",&year,&month);
    sscanf (rec+20," %g %g %g",&hdr.lat,&hdr.lon,&val);
    for (i=0; i<8; i++) hdr.id[i] = rec[i+11];
    /* Time group terminator if needed */
    if (flag) {
      yrsav = year;
      mnsav = month;
      flag = 0;
    }
    if (yrsav!=year || mnsav!=month) {
      hdr.nlev = 0;
      fwrite (&hdr,sizeof(struct rpthdr), 1, ofile);
    }
    yrsav = year;
    mnsav = month;

    /* Write this report */
    hdr.nlev = 1;
    hdr.flag = 1;
    hdr.t = 0.0;
    fwrite (&hdr,sizeof(struct rpthdr), 1, ofile);
    fwrite (&val,sizeof(float), 1, ofile);
  }
  hdr.nlev = 0;
  fwrite (&hdr,sizeof(struct rpthdr), 1, ofile);
}
```

Once the binary data file has been written, create the descriptor file. It would look something like this:

```
dset      rain.dat
dtype     station
stnmap    rain.map
undef     -999.0
title     Rainfall
tdef      12 linear jan1980 1mo
vars      1
  p  0  99  Rainfall
endvars
```

Then run the **stnmap** utility to create the station map file. You can then open and display this data from within GrADS.

5.0 Dimension Environment

The data set is always viewed by GrADS as a generalized 4-D (5-D if you include variables) array located in physical space (lon, lat, lev, time), even if it is in reality a subset of a 4-D space.

The current dimension environment describes what part of the data set you want to work with. Expressions are evaluated with respect to the dimension environment (which allows for simplicity in the expression syntax), and the final display will be determined by the dimension environment. Thus, the dimension environment is a GrADS concept that is important to understand.

The dimension environment is manipulated by the user by entering one of the following **set** commands:

set lat|lon|lev|time val1 <val2>

This set command sets one dimension of the dimension environment using world coordinates.

Alternatively:

set x|y|z|t val1 <val2>

This sets one dimension of the dimension environment using grid coordinates. You may use whatever coordinates are convenient to you. Issuing "**set lon**" is equivalent to issuing "**set x**", both set the x dimension. The difference is only the units you wish to enter the command in.

When you enter just one value, that dimension is said to be "fixed". When you enter two values, that dimension is said to be "varying". The combination of fixed and varying dimensions defines the dimension environment.

Examples:

set lon -180 0	(sets longitude to vary from 180W to 0).
set lat 0 90	(sets latitude to vary from the equator to 90N)
set lev 500	(sets the level to 500mb - a fixed dimension)
set t 1	(sets time to the first time in the data set—using grid coordinates in this case. Time is now a fixed dimension).

When **all** dimensions are fixed, you are referring to a **single** data point.

When **one** dimension is varying, you are referring to a **1-D "slice"** through the data set.

When **two** dimensions are varying, you are referring to a **2-D "slice"** through the data set.

When **three** dimension vary, GrADS interprets this as a **sequence of 2-D slices**.

An important note: When you enter dimensions in grid coordinates, they are always converted to world coordinates. This conversion requires some knowledge of what scaling is in use for grid to world conversions. The scaling that is used in all cases (except one) is the scaling of the *default file*. The exception is when you supply a dimension expression within a variable specification, which will be covered later.

6.0 Variable Names

The complete specification for a variable name is:

abbrev.file#(dimexpr,dimexpr,...) where:

abbrev is the abbreviation for the variable as specified in the data descriptor file
file# is the file number that contains this variable. The default initially is 1. ("**set dfile**" changes the default).
dimexpr is a dimension expression that locally modifies the current dimension environment.

A dimension expression is used to locally modify the dimension environment for that variable only. Only fixed dimensions can be thus modified.

An absolute dimension expression is:

X|Y|Z|T|LON|LAT|LEV|TIME = value

A relative dimension expression (relative to the current dimension environment):

X|Y|Z|T|LON|LAT|LEV|TIME +/- offset

Examples of variable specifications are:

z.3(lev=500)	File 3, absolute dimension expression
tv.1(time-12hr)	Relative dimension expression
rh	Default file number is used
q.2(t-1,lev=850)	Two dimension expressions
z(t+0)	This does have uses....

An important note: When you enter a dimension in grid units, GrADS always converts it to world coordinates. This conversion is done using the scaling of the *default file*. However, when a grid coordinate (x,y,z,t) is supplied within a dimension expression as part of a variable specification, the scaling for that file (ie, the file that variable is to be taken from) is used.

GrADS has a few "**predefined**" variable names. You can think of these as being variables implicitly contained within any opened gridded file. The variable names are:

lat
lon
lev

When used, they will contain the **lat**, **lon**, and **lev** at the respective grid points, using the scaling of the appropriate file. You can specify: **lat.2** for example, to get latitudes on the grid of the 2nd opened data set.

7.0 Expressions

A GrADS expression consists of operators, operands, and parentheses. Parentheses are used the same as in FORTRAN to control the order of operation.

Operators are:

+	Addition
-	Subtraction
*	Multiplication
/	Division

Operands are:

variable specifications, functions, and constants.

Operations are done on equivalent grid points in each grid. Missing data values in either grid give a result of a missing data value at that grid point. Dividing by zero gives a result of a missing data value at that grid point.

Operations cannot be done between grids that have different scaling in their varying dimensions — i.e., grids that have different rules for converting the varying dimensions from grid space to world coordinate space. This can only be encountered when you are attempting operations between grids from different files that have different scaling rules.

If one grid has more varying dimensions than the other, the grid with fewer varying dimensions is 'expanded' and the operation is performed.

Some examples of expressions:

z - z(t-1)	(Height change over time)
t(lev=500)-t(lev=850)	(Temp change between 500 and 850)
ave(z,t=1,t=5)	(Average of z over first 5 times in file)
z - ave(z,lon=0,lon=360,-b)	(Remove zonal mean)
tloop(aave(p,x=1,x=72,y=1,y=46))	(Time series of globally averaged precip, on a 72x46 grid)

8.0 Defined Variables

Defining new variables

The **define** command allows you to interactively create a new variable. The syntax is:

define varname = expr

The new variable can then be used in subsequent expressions (it can be used in subsequent **define** and/or **display** commands). The new variable is stored in memory, not on disk, so avoid defining variables over large dimension ranges.

The variable is defined to cover the dimension ranges in effect at the time the command is issued. You may define a variable that has from 0 to 4 varying dimensions. The **define** command is the only case within GrADS where four varying dimensions is valid.

When Z and/or T are varying dimensions, the **define** command evaluates the expression by stepping through Z and T. In other words, the expression is evaluated within a dimension environment that has fixed Z and T. This will affect how you compose the expression.

When you use a defined variable, data is taken from the variable in a way similar to data taken from a GrADS data file. For example, say you define a four dimensional variable:

set lon -180 0
set lat 0 90
set lev 1000 100
set t 1 10
define temp = rh

After issuing the **define** command, remember to change the dimension environment so less than 4 dimensions are varying!

set t 5
set lev 500
d temp

The display of the defined variable will display a 2-D slice taken at time 5 and level 500.

If you define a variable that has fixed dimensions, and then later access this variable, the fixed dimensions are treated as "wild cards". The best way to show this is with an example:

set lon -180 0
set lat 0 90
set lev 500
set t 10
define zave = ave(z,t=1,t=30)

The defined variable has two varying dimensions. If we now display this variable (or use it in an expression), the fixed dimensions of the defined variable, namely Z and T, will match ANY Z and T dimension setting:

set t 1
set lev 200
d zave

In the above display, the variable **zave** would be displayed as it was defined, ie you would get a time average of 500mb heights, even though the level is set to 850.

When the defined variable has varying dimensions, and you have a dimension environment where that dimension is fixed, the proper dimension will be retrieved from the variable:

set lon -180 0
set lat 0 90
set lev 500
set t 10
define temp = z
set lat 40
d temp

In the above example, the defined variable has a varying Y dimension. We then fix the Y dimension to be 40N, and display a 1-D slice. The data from 40N in the defined grid will be accessed. If you then did:

set lat -40
d temp

The data from 40S would be accessed from the defined variable. Since this is beyond the dimensions originally used when the variable was defined, the data would be set to missing.

You can also locally override the dimension environment:

d temp(lat=50)

If that dimension is a varying dimension within the defined variable. If the dimension is a fixed dimension for that variable, the local override will be ignored:

d temp(t=15)

In the above command, the defined variable temp has fixed T, so the t=15 would be ignored.

Note: the **define** command currently supports only grids.

Once you have defined a grid variables, you may tell GrADS that the new variable is climatological, ie that you wish to treat the time dimension of the new variable in a wild card sense.

The command is:

modify varname <seasonal>
<diurnal>

where the **varname** is the name of the defined grid (the **define** command must have been previously used). If the grid is described as **seasonal**, then it is assumed that the grid contains monthly (or multi-month) means. Note that daily or multi-day means are not yet supported. If **diurnal** is specified, it is assumed the defined variable contains means over some time period less than a day.

After describing the defined variable as climatological, then the date/times are treated appropriately when data is accessed from the defined variable.

An example. The data set contains 10 years of monthly means:

```
set lon -180 180
set lat -90 90
set lev 500
set t 1 12
define zave = ave(z,t+0,t=120,1yr)
```

This define will set up a variable called **zave** which contains 12 times, each time being the 10 year mean for that month. We are making use here of the fact that the **define** command loops through a varying time dimension when evaluating the expression, and within the **ave** function we are making use of the variable time offset of t+0, which uses a start time that is whatever time the **define** command is using as it loops.

```
modify zave seasonal
set t 120
d z - zave
```

The final display will remove the 10 year monthly mean for December from the last December in the data set.

Undefining new variables

Each variable defined using the **define** command reserves some system resources. If you no longer need a defined variable it is sensible to free these resources for other use. This is accomplished with the **undefine** command. For example:

```
undefine p
```

would free the resources used by the defined variable **p**. Of course, the variable **p** would no longer be available for GrADS processing.

9.0 Displaying Data Plots

Displaying your data

The **display** command is how you actually display data (output expressions) plots via the graphics output window. The command is:

display expression

or

d expression

The simplest **expression** is a variable abbreviation.

If you display when **all** dimensions are fixed, you get a **single value** which is typed out.

If you display when **one** dimension varies, you get a **1-D line graph** by default.

If you display when **two** dimensions are varying, you get a **2-D contour plot** by default.

A variety of plot types are available in addition to the above defaults. Choosing these is the subject of the next chapter.

Clearing the Display

GrADS will overlay the output from each display command. To **clear** the display, enter:

clear (or just **c**)

Issued without parameters, the **clear** command does pretty heavy duty clearing of many of the GrADS internal settings. Parameters can be added to limit what is cleared when using more advanced features, for example:

c events	flushes the events buffer (e.g., mouse clicks)
c graphics	clears the graphics, but **not** the widgets
c hbuff	clears the display buffer when in double buffer mode

WARNING: If you make any error in the syntax of **clear** then GrADS does the full **clear**...

10.0 Graphics Output Types

Before you can display a graph of your data, you will need to set the type of plot you want and, probably, some other graphics parameters as well.

By default, when **one** dimension varies, you get a **line graph**, and when **two** dimensions vary, you get a **contour plot**. These defaults can be changed by the command:

set gxout graphics_type

some examples of **graphics_type** are:

contour: Contour plot
shaded: Shaded contour plot
grfill: same as shaded except that each grid box is filled vice contours
grid: Grid boxes with values
vector: Vector wind arrows
stream: Vector streamlines
bar: Bar chart
line: Line Graph
barb: Wind barbs
fgrid: Shaded grid boxes of specified values using **set fgvals ....**
linefill: Color fill between two lines
stat send output to terminal rather than plot

For station data:

value: Station values
barb: Wind barbs
wxsym: Wx symbols
findstn: Find nearest station (see scripting language)
stat send output to terminal rather than plot

There are many options that can be set to control how the **graphics_type** will be displayed. These and other **graphics_types** are covered in detail in Chapter 19 **Graphics Options**.

For the graphics output types of **vector**, **stream**, and **barb**, the display routines need two result grids, where the 1st result grid is treated as the U component, and the 2nd result grid is treated as the V component. To obtain two result grids, you enter two expressions on the display command separated by a semicolon:

display u ; v
display ave(u,t=1,t=10) ; ave(v,t=1,t=10)

For the types of **vector** and **stream**, you can specify a 3rd grid that will be used to colorize the vectors or streamlines:

display u;v;hcurl(u,v)
display u;v;mag(u,v)

For a **gxout** of **wxsym**, each value at a station location is assumed to be a wx symbol code number. Hurricane and tropical storm symbols are included in the symbol set.

draw wxsym symbol x y size <color <thickness>>

Draws the specified **wx** symbol at the specified location. where:

symbol - is an integer specifying what symbol to draw
x - x location, in plotter inches
y - y location
size - size of the symbol (roughly)
color - color of symbol. Use **-1** to get standard colors (red for storm, blue for snow, etc)
thickness - line thickness of the symbol

To see what symbols are available, run grads, then:

run wxsym.gs

You may look at this script to see how to issue the **wxsym** command.

11.0 Animation

If you display when 3 dimensions vary, you get an **animation sequence**. You can animate through any of the three varying dimensions.

By default, the animation dimension is **time**. You may set which dimension to animate through:

set loopdim x|y|z|t

If you wish to animate when fewer than three dimensions are varying (ie, animate a line graph), you can control animation by entering:

set looping on|off

Remember to **set looping off** when you are done animating, or you will get a surprise when you enter your next expression!

12.0 Page Control

Real and virtual pages

The number and size of plots can be controlled on the "real" page by defining one or more "virtual" pages. The relevant command is:

set vpage xmin xmax ymin ymax

This command defines a "virtual page" that fits within the specified limits of the real page. All graphics output will be drawn into this "virtual page" until another **'set vpage'** command is entered. A **clear** command clears the physical page (and any virtual pages drawn on it).

When GrADS is first started, it prompts for landscape or portrait mode. This defines the size of the real page (11x8.5 or 8.5x11). The dimensions for the virtual page must fit within the real page.

The **'set vpage'** command will define virtual page limits in terms of inches (virtual page inches), which are the coordinates that will be used in the various commands that require inches to be used. The new page limits are printed when the **'set vpage'** command completes.

To return to the default state where the real page equals the virtual page, enter:

set vpage off

Controlling the plot area

To control the area within the virtual page that GrADS plots, use:

set parea xmin xmax ymin ymax
off

The command specifies the area for plotting contour plots, maps, or line graphs. This area does not include axis labels, titles, etc., so if you need to see those, provide for an adequate margin.

The region is specified in terms of virtual page units. By default, the virtual page is equal to the real page, so the units are approximately inches on the real page.

Maps are scaled to fit within the plotting area such that their correct aspect ratio is maintained. Thus, the map will not fill the entire plotting area except under certain lat/lon ranges. A line graph or a contour plot without a map will be scaled to fit entirely within the specified plotting area.

By default, an appropriate plotting area is chosen depending on the type of graphics output. To return to this default, enter:

set parea off

It is not appropriate to use this command to put multiple plots on one page. It is better to use the **'set vpage'** command.

13.0 Graphics Primitives

Various commands are provided to allow control and display of various graphics primitives: These enable you to enhance your data plot by adding customised “artwork”. Alternatively, you can use these commands to create, for example, a map-based diagram with no data plot involved.

Drawing commands

draw map

Draw a map outlined as controlled by current settings and the dimension environment.

draw xlab string
ylab

Writes **string** in an appropriate position to label **x** and **y** axes.

draw string x y string

Draws the character **string** at the **x,y** position. **x** and **y** are given in inches on the virtual page. The string is drawn using current string attributes—see the "**set string**" and **"set strsiz**" commands.

draw line x1 y1 x2 y2

Draws a line from **x1, y1** to **x2, y2** using current line drawing attributes. See the "**set line**" command.

draw rec xlo ylo xhi yhi

Draws an unfilled rectangle from **xlo, ylo** to **xhi, ylo** to **xhi, yhi** to **xlo, yhi** to **xlo, ylo**. The rectangle is drawn using current line drawing attributes.

draw recf xlo ylo xhi yhi

Draws a filled rectangle in the area described by **xlo, ylo, xhi, yhi**. The fill color is the current line drawing attribute **color**.

draw mark marktype x y size

Draws a marker of type **marktype** at position **x, y** at the requested **size**. Marker types are:

1 - crosshair
2 - open circle
3 - closed circle
4 - open square
5 - closed square

draw polyf x1 y1 x2 y2 x3 y3 .... xn yn

Draw a filled polygon between a series of **x**,**y** points. The polygon is closed by having **xn** = **x1** and **yn** = **y1**. **Set line** controls the fill color.

Controlling drawing commands

The following commands specify various aspects of the way **draw** commands work.

set font number **where: number = 0 ... 5**

Selects the font for subsequent text operations.

set line color <style> <thickness>

Sets current **line** attributes.

colors are:

0 - black	**5** - cyan
1 - white	**6** - magenta
2 - red	**7** - yellow
3 - green	**8** - orange
4 - blue	**15** - grey

styles are:

1 - solid	**5** - dotted
2 - long dash	**6** - dot dash
3 - short dash	**7** - dot dot dash
4 - long, short dashed	

Thickness values range from **1** to **6**, and provide various line thicknesses on laser printed output.

set string color <justification <thickness <rotation>>>

Sets **string** drawing attributes. **Color** is as described above. **Justification** is the string justification, or how the string is plotted with respect to the x, y position given in the "**draw string**" command. Refer to the following picture for the appropriate codes:

```
 tr            tc               tr          tl - top left
  +-------------+--------------+            tc - center top
  |                            |            tr - right top
l +             + c            + r               etc.
  |                            |
  +-------------+--------------+
 bl            bc              br
```

The **rotation** option specifies the desired string rotation in degrees. When rotated, the center of rotation is the **justification** point. Rotation is counter-clockwise.

set strsiz hsiz <vsiz>

This command sets the string character size, where **hsiz** is the width of the characters, **vsiz** is the height of the characters, in virtual page inches. If **vsiz** is not specified, it will be set the same value as **hsiz**.

set rgb color-number red green blue

Defines new colors within GrADS, and assigns them to a new **color number**. This new **color number** may then be used within any GrADS command that needs a **color number**, such as "**set ccols**".

The **color-number** must be a value between **16** and **99** (**0** to **15** are predefined). The red, green, and blue values must be between **0** and **255**. For example:

set rgb 50 255 255 255

Would define a new **color number**, 50, and assign a color to it. In this case, the color would be white.

The translator **gxps** will make use of the new color settings although the output colors will have to be checked for the desired rendering. **gxps** will *not* directly translate new colors into greyscales—instead, it will translate the *green intensity only* into a new greyscale value. Note that **gxps** has a predefined mapping between color values from 0 to 15 such that the predefined "rainbow" color effect is rendered as a fairly pleasing greyscale gradation, which cannot be done for newly defined colors.

Plot clipping

You may specify a clipping area for drawing graphics primitives such as lines and strings. When you do a **display** command, GrADS sets the clipping region to the **parea**, draws the graphic, then sets the clipping region to the entire page. Even if you have set the clipping region, a display command will reset it to the entire page. To clip the display of the various draw commands:

set clip xlo xhi ylo yhi

where **xlo,xhi,ylo,yhi** are the clipping coordinates in real page inches.

14.0 Hardcopy Output

Producing a GrADS print file

If you plan to do hardcopy output first enter the command:

enable print file-name

This enables the **print** command, and directs print command output to the file given. Any existing contents of this file will be lost.

When you have a graphic displayed that you want to print, enter the command:

print

This will copy the vector instructions used to create the current display into the output file in a GrADS metacode format.

You can close the output file either by quitting GrADS (**quit** command) or the **reinit** command, or by entering:

disable print

Printing a GrADS print file

Once the output file has been closed, the metacode commands within it must be translated to the desired format. The **gxps** utility has been provided to do this. This utility does not run from within the GrADS command environment, you must execute them from the UNIX command line:

gxps - Translate to monochrome postscript (white background). The default GrADS rainbow colors (color numbers 2 to 14) are translated into pleasing greyscale values. User defined colors (numbers above 15) are translated to greyscale intensity based on their *green* content only.

The **gxps** utility converts GrADS graphics meta files (when "printing" in GrADS) to postscript. It solves all conversions problems in one utility using command line options:

-c	color on a *white* background (= old gxpscw)
-r	color on a *black* background (= old gxpsc)
-i fname	where fname is the name of the input GrADS meta file
-o fname	where fname is the name of the output ps file
-d	add a ctrl-d to the end of the file, useful if printing on a HP1200C/PS color printer

The **gxps** utility will prompt for both an input filename and an output filename. The input filename will be the file created by the enable print command. The output filename will be a name of your choice. Any existing file with this name will be deleted. Once the output file is created, you may print it using UNIX print commands. The default is a b/w plot.

15.0 EXEC Command

The **exec** command is used to execute a sequence of GrADS commands from a file. If a **clear** command is encountered, GrADS waits until **enter** is pressed before clearing and continuing with command processing.

The command is:

exec file-name <arg0 arg1 ... arg9>

where: **file-name** is the name of a file containing GrADS commands.

The variables **&0** to **&9** may be used within the exec file to be replaced by blank delimited arguments to the exec.

16.0 Using Station Data

Station Data is also supported within GrADS to a limited extent. Station data consists of data points distributed essentially randomly within the four dimensions.

Operating on station data

Currently, station data operations and display are supported for three distinct dimension environments:

- X, Y varying (horizontal X, Y plot)
- Z varying (vertical profile)
- T varying (time series)

Operations may be done on station data as with gridded data. Operations between grids and station data are not supported.

Operations between station data are defined as being the operation performed on data points that have exactly the same varying dimension values.

For example, if T is the only varying dimension, the expression:

display ts-ds

would result in a time series of station data reports being retrieved for two separate variables. Then, for station reports having exactly the same time, the operation is performed. Note that duplicates are ignored, with the operation being performed between the first occurrences encountered.

When both X and Y are both fixed dimensions, the variable specification may include a station identifier, which specifies a local override for both lat and lon.

The syntax for this would be:

varname(stid=ident)

The station identifiers are case insensitive.

Some functions do not support station data types. These are:

hdivg hcurl vint maskout ave aave tloop

When X and Y are varying, station data values are displayed as numbers centred at their locations. If two expressions are supplied on the **display** command (ie, **display ts;ds**) then two values are displayed, above and below the station location. The display is controlled by the following **set** commands:

set ccolor color
set dignum digits
set digsiz size
set stid on|off

The **'set stid'** command controls whether the station identifier is displayed with each value.

Station Models

GrADs will plot station models from station data. This is enabled by:

set gxout model

The appropriate display command is:

display u;v;t;d;slp;delta;cld;wx;vis

where:

u and **v** are the wind components. A wind barb will be drawn using these values. If either is missing, the station model will not be plotted at all.

t, **d**, **slp**, and **delta** are plotted numerically around the station model:

```
          t    slp
vis wx  O delta
          d
```

cld is the value of the symbol desired at the center of the station model. Values **1** to **9** are assumed to be the marker types (ie, circle, square, crosshair, etc). Values **20** to **25** are assumed to be cloudiness values:

- **20** -clear
- **21** -scattered
- **22** -broken
- **23** -overcast
- **24** -obscured
- **25** -missing (M plotted)

wx is the value of the **wx** symbol (see **'plot wxsym'**) to be plotted in the wx location.

vis is the visibility as a real number. It will be plotted as a whole number and a fraction.

When any of these items are missing (other than **u** and **v**), the model is plotted without that element. To represent a globally missing value, enter a constant in the display command. For example, if the **delta** were always missing, use:

display u;v;t;d;slp;0.0;cld

The station models respond to the usual set commands such as **'set digsiz'**, **'set dignum'**, **'set cthick', 'set ccolor'**.

In addition, there is:

set stnopts <dig3> <nodig3>

which will cause the model to plot the number in the slp location as a three digit number, with only the *last* three digits of the whole number plotted. This allows the standard 3 digit sea level pressure to be plotted by enabling '**dig3**' and plotting slp*10.

17.0 Introduction to GrADS Scripts

Scripts offer users the facility to program GrADS operations. Although it is relatively easy for users to produce sophisticated GrADS graphics without ever writing a script, there are occasions where the programming capability makes things even easier. This chapter explains the general capabilities of scripts, how to run them, and suggests a strategy for users who may wish to write their own.

What scripts can do

The GrADS scripting language, used via the GrADS **run** command, provides a similar capability to the **exec** command, except that a script may have variables, flow control, and access GrADS command output. Scripts may be written to perform a variety of functions, such as allowing a user to point and click on the screen to select something, to animate any desired quantities, to annotate plots with information obtained from GrADS query commands.

The full documentation of the GrADS scripting language is in **Chapter 23 Programming GrADS: Using the Scripting Language**. Before attempting to write your own scripts it is recommended that you read the rest of this chapter and then run some of the supplied scripts (as listed in **Appendix A**). Study the example scripts, referring to **Chapter 23** for information on syntax etc., and you will soon be equipped to write scripts of your own.

Running scripts

The command to execute a script is the **run** command:

run file-name options where: **file-name = *.gs**

This command runs the script contained in the named file.

Automatic script execution

You may have a simple script automatically executed before every **display** command:

set imprun script-name

This script would typically be used to set an option that by default gets reset after each **display** command, for example:

'set grads off'

You can issue any GrADS command from this script, but the interactions are not always clear. For example, if you issued a **display** command from this script, you could easily enter an infinite recursion loop.

The argument to the script is the expression from the display command.

Storing GrADS scripts

It is convenient to put all your GrADS "utility" scripts, such as **cbarn.gs** or **font.gs**, in one directory (e.g., **/usr/local/grads/lib/scripts**).

To simplify running these scripts, GrADS first looks in the current directory for the script and then, if it can't find it, appends the "**.gs**" extension and tries again. For example, suppose you are working on a test script called **t.gs**. You would run it in GrADS by,

run t

If after the first two tries, the script still can't be located, then GrADS looks in the directory defined by the environment variable **GASCRP**. In the **t(csh)**, for example,

setenv GASCRP /home1/grads/lib

or in **ksh**,

export GASCRP=/home1/grads/lib

Note the if the **/** is not added to the end of the directory name, it is automatically added by UNIX. However, it'll still work if you type

setenv GASCRP /home1/grads/lib/

If the script cannot be found, then **.gs** is appended and GrADS tries yet again. Thus,

d slp
run /home1/grads/lib/cbarn.gs

simplifies to,

d slp
run cbarn

18.0 Additional Facilities

Shell commands

Shell commands can be entered at the GrADS command line, by preceding them with an exclamation point:

ga> !ls -l /data/wx/grads

The output of the command, unless redirected, will appear on your console. Shell commands may be executed via a script or exec; but the output of the command will not be returned to the script via the result variable.

Command line options on GrADS utilities

You may specify the file names on the command line of GrADS utilities:

stnmap -i descriptor-name
gxps -i meta-file-name -o ps-file-name

Reinitialisation of GrADS

Two commands have been added to support resetting or reinitializing the state of GrADS:

reset : This command initializes GrADS to its initial state with the following exceptions:

1) No files are closed
2) No defined objects are released
3) The **'set display'** settings are not modified

If files are open, the default file is set to 1, and the dimension environment is set to X,Y varying and Z and T set to 1 (as though file 1 were just opened).

reset can reset only certain parts of GrADS to their initial state by using the following parameters:

reset events	resets the **events** buffer (e.g., mouse clicks)
reset graphics	resets the **graphics**, but **not** the widgets
reset hbuff	resets the **display buffer** when in double buffer mode
reset norset	reset the X events only

reinit: This command does the same as **reset**, and in addition closes all open files and releases all defined objects. It essentially returns GrADS to its initial state just after it is started.

Displaying GrADS Metafiles

A GrADS metafile is the file created with the **print** command which contains instructions for recreating the plot on an output device, such as screen or printer.

gxtran.exe is used to display GrADS meta files. Several command line options are available:

-i fname **fname** is the name of the meta file
-r reverse black and white so you get your plot on white(black) background.
-g WWWWxHHHH+X+Y
This is the same as the command line option in GrADS and allows you to set the geometry of the **x** window as you would in any other X app. The only difference is the space between **-g** and **WWWW**. The display window will be **WWWW** pixels wide **HHHH** pixels tall starting at **X** and **Y** on the screen.
-a animate the frames, i.e., do not pause between frames until the user hits a return in the command window.

For example,

gxtran -r -a -g 800x600+0 -i test.gm

would open a window **800x600** starting at the upper left corner of the screen and animate all frames (plots) in the file **test.gm** using a reverse background.

Reference Section

19.0 Graphics Options

Graphics options control the way graphics output looks. Some options are valid for most graphics output types, some valid for only one.

Some of the options "stick" until changed. Others stay the same until a **clear** command is issued, and yet others are reset to their defaults by either a **clear** command or a **display** command.

1-D Graphics

Line Graphs (gxout = line):

set ccolor color - Sets line color. Reset by **clear** or **display**, where**:**

0 - black	**5** - cyan	**10** - yellow/green	**15** - grey
1 - white	**6** - magenta	**11** - med. blue	
2 - red	**7** - yellow	**12** - dark yellow	
3 - green	**8** - orange	**13** - aqua	
4 - blue	**9** - purple	**14** - dark purple	

set cthick thckns **-** Sets the line thickness for the contours given an integer in the range of **1** to **10** representing relative line thicknesses. Thickness of 6 or more will be thicker on the screen. Primarily used for controlling hardcopy output.

set cstyle style - Sets line style Reset by **clear** or **display**

set cmark marker - Sets line marker:

0 - none
1 - cross
2 - open circle
3 - closed circle
4 - open square
5 - closed square
6 - X
7 - diamond
8 - triangle
9 - none
10 - open circle with vertical line
11 - open oval
Reset by **clear** or **display**

set missconn on - By default, when GrADS plots a line graph, missing data is indicated by a 'break' in the line. set missconn on connects the lines across missing data.

set missconn off - resets the default behavior

Bar Graphs (gxout = bar)

set bargap val **-** Sets the gap between bars in percent. **val** should range from **0** to **100**. The default is **0**, or no gap. A value of **100** gives a single vertical line for each bar.

set barbase val|bottom|top **-** If **val** is given, each bar rises or falls from that value, assuming the value is within the plotting range. If **bottom** is specified, each bar rises from the bottom of the plot. If **top** is specified, each bar falls from the top of the plot.

set baropts opts

where: **opts** =	**outline**	do *not* fill in the bar	
	filled	fill the bar	

set ccolor color - Sets line color. Reset by **clear** or **display**, where:

0 - black	**5** - cyan	**10** - yellow/green	**15** - grey
1 - white	**6** - magenta	**11** - med. blue	
2 - red	**7** - yellow	**12** - dark yellow	
3 - green	**8** - orange	**13** - aqua	
4 - blue	**9** - purple	**14** - dark purple	

set cthick thckns **-** Sets the line thickness for the contours given an integer in the range of **1** to **10** representing relative line thicknesses. Thickness of 6 or more will be thicker on the screen. Primarily used for controlling hardcopy output.

Error Bars (gxout = errbar)

set ccolor color - Sets line color. Reset by **clear** or **display**, where:

0 - black	**5** - cyan	**10** - yellow/green	**15** - grey
1 - white	**6** - magenta	**11** - med. blue	
2 - red	**7** - yellow	**12** - dark yellow	
3 - green	**8** - orange	**13** - aqua	
4 - blue	**9** - purple	**14** - dark purple	

set cthick thckns **-** Sets the line thickness for the contours given an integer in the range of **1** to **10** representing relative line thicknesses. Thickness of 6 or more will be thicker on the screen. Primarily used for controlling hardcopy output.

set bargap val **-** Sets the gap between bars in percent. **val** should range from **0** to **100**. The default is **0**, or no gap. A value of **100** gives a single vertical line for each bar.

set barbase val|bottom|top **-** If **val** is given, each bar rises or falls from that value, assuming the value is within the plotting range. If **bottom** is specified, each bar rises from the bottom of the plot. If **top** is specified, each bar falls from the top of the plot.

Line Graph Shading (gxout = linefill)

set lfcols 1 2
d a;b

Color where **a** < **b** in white (1) and **b** > **a** in red (2)

2-D Gridded Graphics

Line Contour Plots (gxout = contour)

set ccolor color **-** sets the contour color, where: **color** is specified as a color number:

0 - black	**5** - cyan	**10** - yellow/green	**15** - grey
1 - white	**6** - magenta	**11** - med. blue	
2 - red	**7** - yellow	**12** - dark yellow	
3 - green	**8** - orange	**13** - aqua	
4 - blue	**9** - purple	**14** - dark purple	

Reset by **clear** or **display**

You can also issue:

set ccolor rainbow - to obtain the rainbow color sequence.
set ccolor revrain - use the reversed rainbow sequence, complements **set ccolor rainbow**

set cthick thckns - Sets the line thickness for the contours given an integer in the range of **1** to **10** representing relative line thicknesses. Thickness of 6 or more will be thicker on the screen. Primarily used for controlling hardcopy output.

set cstyle style - sets the contour linestyle.
Style numbers are:
1 - solid
2 - long dash
3 - short dash
5 - dots
Reset by **clear** or **display**

set cterp on|off - Turns spline smoothing on or off. "Sticks" until reset. Shaded contours are drawn without spline fitting, so to insure an exact match when overlaying contour lines and shaded contours of the same field, specify **cterp** as **off**. You can still use the **'csmooth'** option, which affects both contour lines and shaded contours.

set cint value - sets the contour interval to the specified value. Reset by a **clear** or **display**
set cmin value - Contours not drawn below this value. Reset by **clear** or **display**.
set cmax value - Contours not drawn above this value. Reset by **clear** or **display**.
set black off/val1 val2 - Contours not drawn within this interval. Reset by **clear** or **display**.
set clevs lev1 lev2 - Sets specified contour levels. Reset by **clear** or **display**
set ccols col1 col2 - Sets specified color for **clev** levels. Reset by **clear** or **display**
(Note: Rainbow sequence is: 9, 14, 4, 11, 5, 13, 3, 10, 7, 12, 8, 2, 6)

set rbrange low high - sets the range of values used to determine which values acquire which rainbow color. By default, the **low** and **high** are set to the min and max of the result grid. This is reset by a **clear** command.

set rbcols color1 color2 <color3> ...
<auto>
Specifies a new **'rainbow'** color sequence. The color numbers may be new colors defined via the **'set rgb'** command. These colors replace the default rainbow color sequence whenever the rainbow color sequence is used. If you **'set rbcols auto'** the built in rainbow sequence is used. This 'sticks' until reset.

set clopts color <thickness <size>>> where:
color - is the label color. **-1** is the default, and indicates to use the contour line color as the label color
thickness - is the label thickness. **-1** is the default.
size - is the label size. **0.09** is the default.
This setting stays set until changed by issuing another **'set clopts'** command.

set csmooth on|off |linear - If on, the grid is interpolated to a finer grid using cubic interpolation before contouring. "Sticks".

Note: this option will result in contour values below and above the min and max of the un-interpolated grid. This may result in physically invalid values such as in the case of negative rainfall. The problem can be avoided by **set csmooth linear**, which uses **linear** interpolation to create the finer grid.

set clab on | off | forced | string | auto **-** Controls contour labelling. "Sticks" until reset.

on - indicates 'fast' contour labels. Labels are plotted where the contour line is horizontal.

off - no contour labels

forced - an attempt is made to label all contour lines

string - specifies a 'substitution' template for conversion of the contour value to character.

This conversion is done by a call to the C system library routine **'sprintf'**. Do a **man** on printf (on the DECstations, do 'man 3s printf') to get information on how this substitution works.

Note that a single floating point number is being passed to the **sprintf** routine for each use of the substitution string.

The result of this substitution is then passed to the GrADS character plotting system. Font controls are also passed, so you may include GrADS font control commands in your substitution string.

Default: **set clab auto** Uses a substitution string of **%g**

Example:

set clab %gK Puts a K on the end of each label
set clab %g%% Puts a percent on the end
set clab %.2f Puts two digits after the decimal point on each label
set clab %03.0f Plots each contour value as a 3 digit integer with leading zeros
set clab Foo Labels each contour with Foo

Warning!!!! No error checking is done on this string! If you specify a multiple substitution (ie, **%g%i**), the **sprintf** routine will look for non-existent arguments and the result will probably be a core dump. You should not use substitutions for types other that float (ie, **%i** or **%s** should not be used).

This option gets reset to '**auto**' when a **clear** is issued.

auto - specifies that you do not want a previously specified string to be used, but instead wish to use the default substitution.

set clskip number **-** where **number** is the number of contour lines to skip when labelling. For example, **set clskip 2** would label every other contour.

Shaded or Grid Fill Contour Plots (gxout = shaded or grfill)

set cint value - sets the contour interval to the specified value. Reset by a **clear** or **display**
set cmin value - Contours not drawn below this value. Reset by **clear** or **display**.
set cmax value - Contours not drawn above this value. Reset by **clear** or **display**.
set black off/val1 val2 - Contours not drawn within this interval. Reset by **clear** or **display**.
set clevs lev1 lev2 - Sets specified contour levels. Reset by **clear** or **display**
set ccols col1 col2 - Sets specified color for **clev** levels. Reset by **clear** or **display**
(Note: Rainbow sequence is: 9, 14, 4, 11, 5, 13, 3, 10, 7, 12, 8, 2, 6)
set rbrange low high **-** sets the range of values used to determine which values acquire which rainbow color. By default, the **low** and **high** are set to the min and max of the result grid. This is reset by a **clear** command.

set rbcols color1 color2 <color3> ...
<auto>

Specifies a new **'rainbow'** color sequence. The color numbers may be new colors defined via the **'set rgb'** command. These colors replace the default rainbow color sequence whenever the rainbow color sequence is used. If you **'set rbcols auto'** the built in rainbow sequence is used. This 'sticks' until reset.

set csmooth on|off |linear - If on, the grid is interpolated to a finer grid using cubic interpolation before contouring. "Sticks".

Note: this option will result in contour values below and above the min and max of the un-interpolated grid. This may result in physically invalid values such as in the case of negative rainfall. The problem can be avoided by **set csmooth linear**, which uses **linear** interpolation to create the finer grid.

Grid Value Plot (gxout = grid)

set dignum number - Number of digits after the decimal place

set digsize size - Size (in inches, or plotter units) of the numbers. **0.1** to **0.15** is usually a good range to use. Both of these options stay the same until changed.

Vector Plot (gxout = vector)

set ccolor color - sets the contour color, where: **color** is specified as a color number:

0 - black	**5** - cyan	**10** - yellow/green	**15** - grey
1 - white	**6** - magenta	**11** - med. blue	
2 - red	**7** - yellow	**12** - dark yellow	
3 - green	**8** - orange	**13** - aqua	
4 - blue	**9** - purple	**14** - dark purple	

Reset by **clear** or **display**

You can also issue:

set ccolor rainbow - to obtain the rainbow color sequence.
set ccolor revrain - use the reversed rainbow sequence, complements **set ccolor rainbow**

set cthick thckns - Sets the line thickness for the contours given an integer in the range of **1** to **10** representing relative line thicknesses. Thickness of 6 or more will be thicker on the screen. Primarily used for controlling hardcopy output.

set arrscl size <magnitude> - Specifies arrow length scaling. **Size** is the length of the arrow in plotting units (inches on the virtual page). A typical value would be **0.5** to **1.0**. **Magnitude** is the vector magnitude that will produce an arrow of the specified size. Other arrow lengths will be scaled appropriately. If **magnitude** is not given, all the arrows will be the same length. Reset by **clear** or **display**.

set arrowhead size - where **size** is the size of the arrowhead. The default is **0.05**. If set to **0**, no arrowhead is plotted. If set to a negative value, the size of the arrowhead will be scaled to the size of the arrow. The value specified will be the size when the arrow is one inch in length.

set cint value - sets the contour interval to the specified value. Reset by a **clear** or **display**

set cmin value - Contours not drawn below this value. Reset by **clear** or **display**.

set cmax value - Contours not drawn above this value. Reset by **clear** or **display**.

set black off/val1 val2 - Contours not drawn within this interval. Reset by **clear** or **display**.

set clevs lev1 lev2 - Sets specified contour levels. Reset by **clear** or **display**
set ccols col1 col2 - Sets specified color for **clev** levels. Reset by **clear** or **display**
(Note: Rainbow sequence is: 9, 14, 4, 11, 5, 13, 3, 10, 7, 12, 8, 2, 6)
set rbrange low high **-** sets the range of values used to determine which values acquire which rainbow color. By default, the **low** and **high** are set to the min and max of the result grid. This is reset by a **clear** command.
set rbcols color1 color2 <color3> ...
<auto>
Specifies a new **'rainbow'** color sequence. The color numbers may be new colors defined via the **'set rgb'** command. These colors replace the default rainbow color sequence whenever the rainbow color sequence is used. If you **'set rbcols auto'** the built in rainbow sequence is used. This 'sticks' until reset.
set arrlab on|off **-** Toggles drawing the vector label for gxout vector: Defaults to **on** and "sticks" (clear doesn't change it's state).

Wind Barb Plot (gxout = barb)

As for gxout = vector.

Scatter Plot (gxout = scatter)

set cmark marker - Sets line marker:
0 - none
1 - cross
2 - open circle
3 - closed circle
4 - open square
5 - closed square
6 - X
7 - diamond
8 - triangle
9 - none
10 - open circle with vertical line
11 - open oval
Reset by clear or display
set digsize size **-** Size (in inches, or plotter units) of the numbers. **0.1** to **0.15** is usually a good range to use. Both of these options stay the same until changed.
set ccolor color - sets the contour color, where: **color** is specified as a color number:

0 - black	**5 -** cyan	**10 -** yellow/green	**15 -** grey
1 - white	**6 -** magenta	**11 -** med. blue	
2 - red	**7 -** yellow	**12 -** dark yellow	
3 - green	**8 -** orange	**13 -** aqua	
4 - blue	**9 -** purple	**14 -** dark purple	

Reset by **clear** or **display**

You can also issue:

set ccolor rainbow - to obtain the rainbow color sequence.
set ccolor revrain - use the reversed rainbow sequence, complements **set ccolor rainbow**

Specific Value Grid Fill Plot (gxout = fgrid)

set fgvals value color <value color> <value color> ... **-** The fgrid output type treats the grid values as rounded integers, and will shade a specified integer valued grid with the specified color. Unspecified values are not shaded. "Sticks".

Streamline Plot (gxout = stream)

set ccolor color - sets the contour color, where: **color** is specified as a color number:

0 - black	**5** - cyan	**10** - yellow/green	**15** - grey
1 - white	**6** - magenta	**11** - med. blue	
2 - red	**7** - yellow	**12** - dark yellow	
3 - green	**8** - orange	**13** - aqua	
4 - blue	**9** - purple	**14** - dark purple	

Reset by **clear** or **display**

You can also issue:

set ccolor rainbow - to obtain the rainbow color sequence.
set ccolor revrain - use the reversed rainbow sequence, complements **set ccolor rainbow**

set cint value - sets the contour interval to the specified value. Reset by a **clear** or **display**
set cmin value - Contours not drawn below this value. Reset by **clear** or **display**.
set cthick thckns **-** Sets the line thickness for the contours given an integer in the range of **1** to **10** representing relative line thicknesses. Thickness of 6 or more will be thicker on the screen. Primarily used for controlling hardcopy output.
set cmax value - Contours not drawn above this value. Reset by **clear** or **display**.
set black off/val1 val2 - Contours not drawn within this interval. Reset by **clear** or **display**.
set clevs lev1 lev2 - Sets specified contour levels. Reset by **clear** or **display**
set ccols col1 col2 - Sets specified color for **clev** levels. Reset by **clear** or **display**
(Note: Rainbow sequence is: 9, 14, 4, 11, 5, 13, 3, 10, 7, 12, 8, 2, 6)
set rbrange low high **-** sets the range of values used to determine which values acquire which rainbow color. By default, the **low** and **high** are set to the min and max of the result grid. This is reset by a **clear** command.
set rbcols color1 color2 <color3> ... <auto>
Specifies a new **’rainbow’** color sequence. The color numbers may be new colors defined via the **’set rgb’** command. These colors replace the default rainbow color sequence whenever the rainbow color sequence is used. If you **’set rbcols auto’** the built in rainbow sequence is used. This ’sticks’ until reset.
set strmden value **-** specifies the streamline density, where the value is from **1** to **10**. **5** is default.

1-D Station Graphics

Plot time series of wind barbs at a point (gxout = tserbarb)

For example:

set gxout tserbarb
d us(stid=79001;vs(stid=79001)

Plot time series of weather symbols at a point (gxout = tserwx)

For example:

set parea 0.75 10.5 3.875 4.25
set grads off
set gxout tserwx
set digsiz 0.11
d wx(stid=79001)

where the grid **wx** contains codes for weather symbols

2-D Station Graphics

Plot station values (gxout = value)

set ccolor color - sets the contour color, where: **color** is specified as a color number:

0 - black	**5** - cyan	**10** - yellow/green	**15** - grey
1 - white	**6** - magenta	**11** - med. blue	
2 - red	**7** - yellow	**12** - dark yellow	
3 - green	**8** - orange	**13** - aqua	
4 - blue	**9** - purple	**14** - dark purple	

Reset by **clear** or **display**

You can also issue:

set ccolor rainbow - to obtain the rainbow color sequence.
set ccolor revrain - use the reversed rainbow sequence, complements **set ccolor rainbow**

set cthick thckns **-** Sets the line thickness for the contours given an integer in the range of **1** to **10** representing relative line thicknesses. Thickness of 6 or more will be thicker on the screen. Primarily used for controlling hardcopy output.
set digsize size **-** Size (in inches, or plotter units) of the numbers. **0.1** to **0.15** is usually a good range to use. Both of these options stay the same until changed.
set stid on/off **-** Controls whether the **station id** is displayed next to the values or not.

Plot wind barb at station (gxout = barb)

set ccolor color - sets the contour color, where: **color** is specified as a color number:

0 - black	**5** - cyan	**10** - yellow/green	**15** - grey
1 - white	**6** - magenta	**11** - med. blue	

2 - red	**7** - yellow	**12** - dark yellow	
3 - green	**8** - orange	**13** - aqua	
4 - blue	**9** - purple	**14** - dark purple	

Reset by **clear** or **display**

You can also issue:

set ccolor rainbow - to obtain the rainbow color sequence.
set ccolor revrain - use the reversed rainbow sequence, complements **set ccolor rainbow**

set cthick thckns - Sets the line thickness for the contours given an integer in the range of **1** to **10** representing relative line thicknesses. Thickness of 6 or more will be thicker on the screen. Primarily used for controlling hardcopy output.
set digsize size - Size (in inches, or plotter units) of the numbers. **0.1** to **0.15** is usually a good range to use. Both of these options stay the same until changed.

Plot weather symbol at station (gxout = wxsym)

set ccolor color - sets the contour color, where: **color** is specified as a color number:

0 - black	**5** - cyan	**10** - yellow/green	**15** - grey
1 - white	**6** - magenta	**11** - med. blue	
2 - red	**7** - yellow	**12** - dark yellow	
3 - green	**8** - orange	**13** - aqua	
4 - blue	**9** - purple	**14** - dark purple	

Reset by **clear** or **display**

You can also issue:

set ccolor rainbow - to obtain the rainbow color sequence.
set ccolor revrain - use the reversed rainbow sequence, complements **set ccolor rainbow**

set cthick thckns - Sets the line thickness for the contours given an integer in the range of **1** to **10** representing relative line thicknesses. Thickness of 6 or more will be thicker on the screen. Primarily used for controlling hardcopy output.
set digsize size - Size (in inches, or plotter units) of the numbers. **0.1** to **0.15** is usually a good range to use. Both of these options stay the same until changed.
set wxcols c1 c2 c3 c4 c5 c6 - where: **c1 ... c6** are the symbol colors

Plot station model (gxout = model)

set ccolor color - sets the contour color, where: **color** is specified as a color number:

0 - black	**5** - cyan	**10** - yellow/green	**15** - grey
1 - white	**6** - magenta	**11** - med. blue	
2 - red	**7** - yellow	**12** - dark yellow	
3 - green	**8** - orange	**13** - aqua	
4 - blue	**9** - purple	**14** - dark purple	

Reset by **clear** or **display**

You can also issue:

set ccolor rainbow - to obtain the rainbow color sequence.

set ccolor revrain - use the reversed rainbow sequence, complements **set ccolor rainbow**

set cthick thckns - Sets the line thickness for the contours given an integer in the range of **1** to **10** representing relative line thicknesses. Thickness of 6 or more will be thicker on the screen. Primarily used for controlling hardcopy output.
set digsize size - Size (in inches, or plotter units) of the numbers. **0.1** to **0.15** is usually a good range to use. Both of these options stay the same until changed.
set mdlopts opt - where: **opt** = noblank, blank, dig3, or nodig3
set wxcols c1 c2 c3 c4 c5 c6 - where: **c1 ... c6** are the symbol colors

Other Display Options

Find closest station to x,y point (gxout = findstn)

See Chapter 23 Programming GrADS.

Write data to file (gxout = fwrite)

set fwrite out.dat
set gxout fwrite

Output the grid to the file **out.dat** (goes to **grads.fwrite** by default). The data consist of local floats (e.g. 32-bit IEEE big endian floats on a sun/sgi/hp workstation)

Display information about data (gxout = stat)

set gxout stat
d

sends output to the terminal as opposed to a plot or data output (e.g., **set fwrite out.dat** ; **set gxout fwrite; d rh**). Or the output goes to the script variable "**result**" which can be parsed inside a script (see the **corr.gs** GrADS script)

The output allows many statistical calculations to be made. Here's an example of opening up a global model file and looking at the 1000 mb relative humidity, statistically,

```
ga-> set gxout stat
ga-> d rh
Data Type = grid
Dimensions = 0 1
I Dimension = 1 to 145
J Dimension = 1 to 73
Sizes = 145 73 10585
Undef value = 1e+20
Undef count = 0  Valid count = 10585
Min, Max = 0.0610352 100.061
Stats(sum,sumsqr,n):     787381 6.35439e+07 10585
Stats(sum,sumsqr)/n:     74.3865 6003.2
Stats(sum,sumsqr)/(n-1): 74.3935 6003.77
Stats(sigma,var)(n):     21.6761 469.854
Stats(sigma,var)(n-1):   21.6771 469.898
```

Cmin, cmax, cint = 10 100 10

Let's break it down:

Data Type = grid ----- you have a grid
Dimensions = 0 1 ----- the dimension type for the variable
0 - lon
1 - lat
2 - lev
3 - time
1 - not varying
I Dimension = 1 to 145 ------ obvious
J Dimension = 1 to 73
Sizes = 145 73 10585 ------- **10585** is 145*73 or total number of points
Undef value = 1e+20 ------- undefined value
Undef count = 0 Valid count = 10585 ----- # of defined and undefined points in the grid. Remember that if GrADS can't find any data it returns undefs. This is useful for checking if you have any data, **Valid count = 0** means no...
Min, Max = 0.0610352 100.061 ---- UHHH OHHHH! we have slight supersaturation..
Stats(sum,sumsqr,n): 787381 6.35439e+07 10585 - This should be fairly obvious, **sum =** the simple sum of all **defined** points. **sumsqr =** sum of, in this case, **rh*rh** and **10585** is **n**.
Stats(sum,sumsqr)/n: 74.3865 6003.2 - Divide by **n** for convenience, the first number is the "biased" mean...
Stats(sum,sumsqr)/(n-1): 74.3935 6003.77 - the so called "**unbiased**" mean (remove 1 degree of freedom), etc.
Stats(sigma,var)(n): 21.6761 469.854 - the standard deviation and variance "biased" (**n**)
Stats(sigma,var)(n-1): 21.6771 469.898 - the standard deviation and variance "unbiased" (**n-1**)
Cmin, cmax, cint = 10 100 10 - What GrADS will use when contouring.

NOTE: This works for both **gridded** and **station** data

Set Commands to Control Graphics Display

Set range for plotting 1-D or scatter plots

set vrange y1 y2 - Specifies the range of the variable values for y-axis scaling (from y1 - y2). Reset by **clear** only.
set vrange2 x1 x2 - Specifies the range of the variable values for x-axis scaling (from x1 - x2). Reset by **clear** only.

To control log scaling when the Z dimension is plotted on any plot:

set zlog on|off - Sets log scaling of the Z dimension **on** or **off**. Sticks until **reset**.

To control axis orientation:

set xyrev on off - Reverses the axes on a plot.

Example:

By default for a Z, T plot, the time dimension is plotted horizontally, and the Z dimension is plotted vertically. By setting **xyrev**, the time dimension would be plotted vertically and the Z dimension would be plotted horizontally.

set xflip on off - flips the order of the horizontal axis (whatever axis is horizontal after xyrev is applied).
set yflip on off - flips the order of the vertical axis.

All the above axis orientation commands are reset by a ’**clear**’ or **’set vpage’** command.

To control axis labelling

set xaxis|yaxis start end <incr> - Specifies the axis is to be labelled from the specified **start** value to the specified **end** value with the specified **increment**. Labels may have no relation to data or dimensions.
set xlint interval
ylint

Specifies the label **interval**. Overridden by **’set xlevs/ylevs’**. Reset by a **clear** command.

Note that labels are always ’started’ at **0**. ie, if you set the interval to **3**, the labels will be **0, 3, 6, 9** ...

If you set the **interval** to a negative value, this indicates to start the labelling at the axis start value. If this were **1** (with and interval of **3**), then the labels would be **1, 4, 7**...

This setting will override a labelling interval specified on the **xaxis** or **yaxis** commands.

This command does not apply to a date/time axis.

set xlab on | off | auto | string
ylab

This works the same way as the **’set clab’** command. An example:

set ylab %.2f

would plot all the Y axis labels with 2 digits after the decimal point. This is reset by a **clear** command.

set xlevs lab1 lab2 ...
ylevs

Specifies the label **levels** to plot for the X or Y axis. Each label desired is specified by the user. Reset by a **clear** command.

If you have **set xaxis** or **yaxis**, then **set xlevs/ylevs**, the levels specified apply to the labelling range specified in the **xaxis** or **yaxis** command.

This command does not apply to a date/time axis.

set xlopts color <thickness < size >> - controls aspects of axis display, where:
ylopts

xlopts controls the X Axis
ylopts controls the Y Axis
color: Label color (Default 1)
thickness: Label thickness (Default 4)
size: Label size (Default 0.12)

set xlpos offset side - controls position of x axis labels, where:
offset offset in inches
side **b** or **t** (bottom or top)

set ylpos offset side - Controls position of y axis labels, where:
offset offset in inches
side **r** or **l** (right or left)

To control displayed map projections

set mproj proj - Sets current map projection, keywords are:

robinson Robinson projection, requires set lon -180 180, set lat -90 90
latlon the default. Lat/lon projection with aspect ratio maintained.
scaled latlon projection where aspect ratio is not maintained. The plot fills the plotting area.
nps north polar stereographic
sps south polar stereographic
off same as scaled, but no map is drawn and axis labels are not interpreted as lat/lon labels.

set mpvals lonmin lonmax latmin latmax
off
Sets reference longitudes and latitudes for polar stereographic plots. By default, these are set to the current dimension environment limits. This command overrides that, and allows the data-reference to be decoupled with the map display. The polar plot will be drawn such that the region bounded by these longitudes and latitudes will be entirely included in the plot.

GrADS will plot lat/lon lines on polar plots with no labels as yet. To turn this off,

set grid off

To control map drawing:

set mpdset <lowres|mres|hires|nmap> - selects base map resolution, where:
lowres is the default
mres and **hires** have state and country outlines
nmap covers only North America.

set poli on|off - Selects whether you want political boundaries drawn for the **mres** or **hires** map data sets. The default is **on**.

set map auto color style thickness - Draws the map background using the requested line attributes.

set mpdraw on|off - If **off**, does not draw the map background. Requested map scaling is still in force.

set grid on | off | style values | horizontal | vertical - Draw or do not draw grid lines using the specified **color** and **linestyle**. Default is to draw grid lines with color **15** (grey) and with linestyle **5** (dotted). **horizontal** indicates to only plot the horizontal grid lines; **vertical** the vertical grid lines.

All the above settings stay the same until changed by new **set** commands.

To control annotation

set font number - Selects the font for subsequent text operations, where: **number = 0 ... 9**.
draw title string - Draw **title** at top of graph. Backslash within string denotes new line.
set annot color <thickness> - Sets **color** and line **thicknesses** for the above 3 draw commands. Default is **white**, thickness **6**. This command also sets the **color** and **thickness** for the axis border, axis labels, and tickmarks. Axis tickmarks and labels are plotted at the specified thickness minus 1.

To control console display

set display grey|greyscale|color <black|white> - Sets the mode of the display.

By default, the mode is **color**, where shading and contouring is done with a rainbow of colors. When using a monochrome display, these colors may not map to greyscale in a pleasing way. When the mode is set to **greyscale**, contours are displayed using a single grey level, and shaded contours are done using a sequence of greyscales.

You may optionally set the hardware background color to **black** or **white**. The default is **black**.

set display grey white

gives a result on the display that is very similar to the output produced by **gxps**.

This command DOES NOT affect hardcopy output.

set background white/black the default is black

To control the frame

set frame on | off | circle - where:

on plots a rectangular frame around the clipped region
off plots no frame
circle plots a rectangular frame for lat-lon projections, plots a circular frame for a polar plot at the outermost latitude. Used for whole-hemisphere plots only.

To control logo display

set grads on|off **- off** disables display of the GrADS logo from the screen and printed output. Reset by **clear**.

20.0 GrADS Functions

The real power of GrADS lies in its data analysis capabilities. These capabilities are accessed through GrADS internal or user defined external functions (the so-called **udf**s, see Chapter 21) applied to an expression (GrADS talk for a grid of data, e.g. var.1(t=1)).

Functions are invoked by name, with their arguments separated by commas and enclosed in parentheses.

Expressions are typically one of the arguments supplied to a function. Functions may be nested. Some functions modify the dimension environment when they operate.

The following list of GrADS functions is grouped alphabetically under descriptive categories.

Averaging Functions

aave

aave(expr,xdim1,xdim2,ydim1,ydim2)

Takes an area average over an X-Y region. Usually better than using nested **ave** functions.

expr	any valid GrADS expression.
xdim1	starting dimension expression for the X dimension.
xdim2	ending dimension expression for the X dimension.
ydim1	starting dimension expression for the Y dimension.
ydim2	ending dimension expression for the Y dimension.

Usage Notes

1) In the absence of missing data values, **aave** gives the same result as nested **ave** functions in the X and Y dimensions:

   ave(ave(*expr*,x=1,x=72),y=1,y=46)

   being equivalent to:

   aave(*expr*,x=1,x=72,y=1,y=46)

   in terms of the numerical result. The **aave** function is more efficient.

2) When there are missing data values, the **aave** function does not return the same result as nested **ave** functions. To see this, consider the small grid:

6	18	3	5
10	10	10	10
12	U	U	U

   where U represents the missing data value. If we apply nested **ave** functions, the inner **ave** will provide row averages of 8, 10, and 12. When the outside **ave** is applied, the result will be an average of 10. When **aave** is used, all the values participate equally (in this case, we are assuming no weights applied to the final average), and the result is 84/9 or about 9.33.

3) The **aave** function assumes that the world coordinates are longitude in the X dimension and latitude in the Y dimension, and does weighting in the latitude dimension by the delta of the sin of the latitudes. Weighting is also performed appropriately for unequally spaced grids.

4) The **aave** function always does its average to the exact boundaries specified, in world coordinates. This is somewhat different from the **ave** function, where the **-b** flag is used to get this behavior. If the boundaries specified via the dimension expressions do not fall on grid boundaries, then the boundary values are weighted appropriately in the average.

When grid coordinates are used in the dimensions expressions, then they are converted to world coordinates for the boundary to be determined. This conversion is done using the scaling of the default file. Note that the conversion is done using the outside grid boundary, rather than the grid center. For example:

aave(expr,x=1,x=72,y=1,y=46)

Here the boundary would be determined by using the grid values 0.5, 72.5, 0.5, and 46.5 which would be converted to world coordinates. If we assume that x=1 is 0 degrees longitude and x=72 is 355 degrees longitude, then the averaging boundary would be -2.5 to 357.5 degrees, which would cover the earth. In the Y dimension, when the boundary is beyond the pole, the **aave** function recognizes this and weights appropriately.

Examples

1) See the **tloop** function for an example of creating a time series of area averages.
2) An example of taking an area average of data only over land, given a mask grid:

aave(maskout(p,mask.3(t=1)),x=1,x=72,y=1,y=46)

In this case, it is assumed the mask grid has negative values at ocean points.

amean

amean(expr,xdim1,xdim2,ydim1,ydim2)

Works exactly like **aave**, except that area weighting is disabled, i.e., the mean is a straight sum / # points.

ave

ave(expr,dexpr1,dexpr2<,tincr<,flags>>)

Averages the result of ***expr*** over the specified range of dimensions starting at ***dexpr1*** and ending at ***dexpr2*.** If the averaging dimension is time, an optional time increment ***tincr*** may be specified.

expr is any valid GrADS expression.
dexpr1 is the start point for the average, specified as a standard GrADS dimension expression.
dexpr2 is the end point for the average. The dimensions of ***dexpr1*** and ***dexpr2*** must match.
tincr is a time increment for the average, if ***dexpr1*** and ***dexpr2*** are time dimension.
flags The following flags are valid:

b The ***boundary*** flag indicates the average should be taken to the exact boundaries specified in ***dexpr1*** and ***dexpr2***, rather than nearest grid points. See *Usage Notes* below.

Usage Notes

1) The limits and interval over which to take the average are determined by the scaling of the default file. Conversions of ***dexpr1*** and ***dexpr2*** to grid coordinates are performed using the scaling of the default file. See the *Examples* below for an example of what this means.

2) The average is weighted for non-linear grid intervals. Averages over latitude are weighted by the delta of the sine of the latitudes at the edge of the grid box. The edges of the grid box are always defined as being the midpoint between adjacent grid points.

3) If ***dexpr1*** and ***dexpr2*** are specified in world coordinates, the coordinates are converted to the nearest integer grid coordinates based on the scaling of the default file. The average is then performed over the range of these grid coordinates. The end points are given normal weighting, unless the **-b** flag is specified.

Examples

All the examples use the two example descriptor files given at the beginning of this section. For the following examples, the dimension environment is X-Y varying; Z-T are fixed.

1) Consider the following average, when the default file is file #1:

   ave(z.2,t=1,t=10)

   We are averaging a variable from file #2, but using the scaling from file #1. File #1 has a time interval of 6 hours, but file #2 has a time interval of 12 hours. The average will thus attempt to access data from file #2 for times that are not available, and an error will occur. To avoid this, the default file should be set to file #2: **set dfile 2**

2) The average:

   ave(z,t=1,t=120,4)

   will average only 00Z reports from file #1, since the time increment is 4, which for this file is 24 hours.

3) If you attempt to take a zonal average as follows:

   ave(z,lon=0,lon=360)

   the world coordinates will be converted to grid coordinates, here X varying from 1 to 181, and the grid point at longitude 0 (and 360) will be used twice in the average. To have the end points of this average weighted properly, use the **-b** flag:

   ave(z,lon=0,lon=360,-b)

   or average using the grid coordinates directly:

   ave(z,x=1,x=180)

4) You can nest averaging operations:

   ave(ave(z,x=1,x=180),y=1,y=46)

   In this case, to take an areal average. Note that for areal averaging, the **aave** function is better. See the **aave** function description.

   When nesting averages, the order of the nesting can have a dramatic affect on performance. Keep in mind the ordering of the data in a GrADS file: X varies the fastest, then Y, then Z, then T. When nesting averages, put the faster varying dimension within the inner average:

```
set lon -90
set lat -90 90
set lev 1000 100
d ave(ave(t,x=1,x=180),t=1,t=20)
```

This average would be more efficient than, for example:

```
ave(ave(t,t=1,t=20),x=1,x=180)
```

although the final numerical result would be the same.

5) The use of the **define** command can make certain operations much more efficient. If you want to calculate standard deviation, for example:

```
sqrt(ave(pow(ave(z,t=1,t=20)-z,2),t=1,t=20))
```

would be correct, but the inside average would be calculated 20 times. Defining the inside average in advance will be substantially faster:

```
define zave = ave(z,t=1,t=20)
d sqrt(ave(pow(zave-z,2),t=1,t=20))
```

mean

mean(expr,dexpr1,dexpr2<,tincr<,flags>>)

Works exactly like **ave**, except that area weighting is disabled, i.e., the mean is a straight sum / # points.

vint

vint(expr,psexpr,top)

Performs a mass-weighted vertical integral in mb pressure coordinates, where:

expr A GrADS expression for the quantity to be integrated.
psexpr An expression yielding the surface pressure, in mb, which will be used to bound the integration on the bottom.
top A constant, giving the bounding top pressure, in mb. This value cannot be provided as an expression.

The calculation is a sum of the mass-weighted layers:

$$\frac{1}{g}\sum expr\Delta p\ldots$$

where the bounds are the surface pressure on the bottom and the indicated ***top*** value on the top. The summation is done for each layer present that is between the bounds. The layers are determined by the different levels of the Z dimension from the default file. Each layer is considered to be from the midpoints between the levels actually present, and is assumed to have the same value throughout the layer, namely the value of the gridpoint at the middle of the layer.

Usage Notes

1) Since the summation is done using the Z levels from the default file, it is important that the default file have the same Z dimension coordinates as the ***expr***.

2) Levels of data below and above the bounds of the summation (surface pressure on bottom; ***top*** value on the top) are ignored, even if present.
3) It is assumed the world dimension values for the Z dimension are mb pressure values. The units of *g* are such that when the expression integrated is specific humidity (q) in units of g/g, the result is g of water per square meter, or essentially precipitable water in mm.
4) It is usually a good idea to make the ***top*** pressure value to be at the top of a layer. For example, if the default file (and the data) have pressure levels of ...,500,400,300,250,... then a good value for ***top*** might be 275, the value at the top of the layer that extends from 350 to 275 mb.
5) The **vint** function operates only in an X-Y varying dimension environment.

Examples

1) A typical use of **vint** might be:

   vint(q,ps,275)

   to integrate specific humidity to obtain precipitable water, in mm.

Filtering Functions

smth9

smth9(*expr*)

Performs a 9 point smoothing to the gridded result of the *expr*.

Usage Notes

1) The result at each grid point is a weighted average of the grid point plus the 8 surrounding points. The center point receives a weight of 1.0, the points at each side and above and below receive a weight of 0.5, and corner points receive a weight of 0.3.
2) All 9 points are multiplied by their weights and summed, then divided by the total weight to obtain the smoothed value. Any missing data points are not included in the sum; points beyond the grid boundary are considered to be missing. Thus the final result may be the result of an averaging with less than 9 points.
3) If the gridded data is 1-Dimensional, the result is a 3 point smoothing.

Finite Difference Functions

cdiff

cdiff(expr,dim)

Performs a centred difference operation on ***expr*** in the direction specified by ***dim***. The difference is done in the grid space, and no adjustment is performed for unequally spaced grids. The result value at each grid point is the value at the grid point plus one minus the value at the grid point minus one. The ***dim*** argument specifies the dimension over which the difference is to be taken, and is a single character: X, Y, Z, or T.

Result values at the grid boundaries are set to missing.

Examples

1) The **cdiff** function may be used to duplicate the calculation done by the hcurl function:

   define dv = cdiff(v,x)
   define dx = cdiff(lon,x)*3.1416/180
   define du = cdiff(u*cos(lat*3.1416/180),y)
   define dy = cdiff(lat,y)*3.1416/180
   display (dv/dx-du/dy)/(6.37e6*cos(lat*3.1416/180))

   The above example assumes an X-Y varying dimension environment. Note that the intrinsic variables **lat** and **lon** give results in degrees and must be converted to radians in the calculation. Also note the radius of the earth is assumed to be 6.37e6 meters thus the U and V winds are assumed to have units of m/sec.

2) Temperature advection can be calculated using the **cdiff** function as follows:

   define dtx = cdiff(t,x)
   define dty = cdiff(t,y)
   define dx = cdiff(lon,x)*3.1416/180
   define dy = cdiff(lat,y)*3.1416/180
   display -1*((u*dtx)/(cos(lat*3.1416/180)*dx) + v*dty/dy)/6.37e6

   where the variable **t** is temperature, **u** is the U component of the wind, and **v** is the V component of the wind.

Grid Functions

const

const(expr,constant<,flag>)

Some or all of the values in ***expr*** are set to the ***constant*** value, depending on the value of ***flag***:

expr A valid GrADS expression.
constant

A constant, given as an integer or floating point value. The value will be treated as floating point.

flag If no flag is specified, all the non-missing data values in ***expr*** are set to the ***constant*** value to form the result. Missing data values are preserved as missing.

Other flag values:

u All missing data values are set to the ***constant*** value. Non-missing data remains unchanged.
a All possible data values in the result are set to the ***constant*** value, including missing data values.

Usage Notes

1) This function operates on both gridded and station data.

Examples

1) The **const** function may be used to assign a new value to missing data, so that missing data may participate in operations:

   const(z,0.0,-u)

2) The **const** function is useful when displaying plots using the **linefill** graphics output type when one of the lines needs to be a straight horizontal line:

   set lon -90
   set lat -90 90
   set gxout linefill
   set lev 500
   display const(t,-20);t-273.16

3) The **const** function may be used to calculate the fraction of the globe covered by some value of interest. In this case, the portion of the globe covered by precipitation greater than 10 mm/day is calculated as a time series:

   set lon 0 360
   set lat -90 90
   set t 1 last
   define ones = const(const(maskout(p,p-10),1),0,-u)
   set x 1
   set y 1
   display tloop(aave(ones,lon=0,lon=360,lat=0,lat=360))

   Note that we first create a defined array that contains 1 wherever the precip value is greater than 10, and 0 whenever the precip value is less than 10. This is done via nested functions, where we first use the **maskout** function to set all values less than 10 to missing. We then use a **const** function with no arguments to set all non-missing values to 1, then use a **const** function with the **-u** flag to set all the missing data values to 0. The **aave** function is used to calculate an area weighted average. Since we are averaging zeros and ones, the result is the fraction of the area where there are ones. See the **tloop** function for a description of how to perform time series of areal averages.

maskout

maskout(expr,mask)

Wherever the ***mask*** values are less than zero, the values in ***expr*** are set to the missing data value.

Works with gridded or station data. Where ***mask*** values are positive, the ***expr*** values are not modified. Thus the result of **maskout** is data with a possibly increased number of missing data values. The **maskout** function, in spite of its apparent simplicity, is extremely useful.

Examples

1) See the Examples for the **const** function for a description of using **maskout** to calculate the percentage of the globe covered by precipitation.

2) The **maskout** function can be used to cause part of the data to be ignored while doing another calculation. For example, if we have a land-sea mask, where sea values are negative, and we want to take some areal average of a quantity only over land:

   d aave(maskout(p,mask.2),lon=0,lon=360,lat=0,lat=90)

3) People frequently have trouble using a mask grid, because it is often put into a separate file, and given some arbitrary date/time and level. Thus, it is often necessary to *locally override* the dimension environment while using the mask grid:

   d aave(maskout(p,mask.2(t=1)),lon=0,lon=360,lat=0,lat=90)

   would probably be how Example 2 would have to be expressed in order to work, with the local override of **t=1** specified on the mask data. See the documentation on how GrADS evaluates expressions within the dimension environment for more information.

skip

skip(expr,skipx,skipy)

Sets alternating values of the ***expr*** to the missing data value. This function is used while displaying wind **arrows** or **barbs** to thin the number of arrows or barbs.

expr	A grid expression.
skipx	Skip factor in the X direction.
skipy	Number of grid points to skip in the Y direction.

Examples

1) To display every other grid point in both the X and Y direction:

   d skip(u,2,2);v

2) To display every grid point in the X direction, but every 5^{th} grid point in the Y direction:

   d skip(u,1,5);v

   Note that it is not necessary to use the **skip** function on both the U and V wind components; it is sufficient to populate only one component with missing data values to suppress the plotting of the wind **arrow** or **barb**.

Math Functions

abs

abs(*expr*)

Takes the absolute value of the result of ***expr***. Operates on both gridded and station data. Missing data values do not participate.

acos

acos(*expr*)

Applies the $\cos^{-1}$ function to the result of ***expr***. Values from ***expr*** that exceed 1 or are less than -1 are set to missing. The result of the **acos** function is in radians.

asin

asin(*expr*)

Applies the $\sin^{-1}$ function to the result of ***expr***. Values of ***expr*** that exceed 1 or are less than -1 are set to missing in the final result. The result of the **asin** function is in radians.

atan2

atan2(expr1,expr2)

Applies the $\tan^{-1}$ function to the result of the two expressions, using the formula:

$$\tan\theta = \frac{y}{x}$$

where y is ***expr1*** and x is ***expr2***. Situations where x and y are both zero are valid; the result is arbitrarily set to zero. The result of the **atan** function is in radians.

cos

cos(*expr*)

Takes the cosine of the ***expr***. Values are assumed to be in radians. Works on both gridded and station data.

exp

exp(*expr*)

Performs the e**x operation, where ***expr*** is x. Works on both gridded and station data.

gint

gint(expr)

General integral, same as ave except do not divide by the total area

log

log(*expr*)

Takes the natural logarithm of the expression. May be used with gridded or station data. Values less than or equal to zero are set to missing in the result.

log10

log10(*expr*)

Takes the logarithm base 10 of the expression. May be used with gridded or station data. Values less than or equal to zero are set to missing in the result.

pow

pow(expr1,expr2)

The **pow** function raises the values of ***expr1*** to the power of ***expr2***. No error checking is performed for invalid values in the operands. This function works on both gridded and station data.

Examples

1) To square some value:

    pow(expr,2)

2) To duplicate the operation of the mag function:

    sqrt(pow(u,2)+pow(v,2))

sin

sin(*expr*)

Takes the sin of the provided expression. It is assumed the expression is in radians. Result values are in the range -1 to 1. This function works on both gridded and station data.

sqrt

sqrt(*expr*)

Takes the square root of the result of the ***expr***. This function works on both gridded and station data. Values in ***expr*** that are less than zero are set to missing in the result.

tan

tan(*expr*)

Applies the trigonometric tangent function to the ***expr*** which is assumed to be in radians. Operates on both gridded and station data.

Meteorological Functions

tvrh2q

tvrh2q(tvexpr,rhexpr)

Given virtual temperature and relative humidity, **tvrh2q** returns specific humidity, q, in g/g. Specifically:

tvexpr	A valid GrADS expression where the result is virtual temperature, in Kelvin.
rhexpr	A GrADS expression that results in relative humidity, in percent (from 0 to 100).

This function works only on gridded data.

Usage Notes

1) The conversion formula requires a pressure in mb. **tvrh2q** assumes that the Z coordinate system is pressure in mb. If Z is a varying dimension, the pressure valid at each grid point is used. When Z is a fixed dimension, the Z value from the current dimension environment is used.

   Note that it is possible to provide values from an incorrect pressure level by overriding the current dimension environment:

   set lev 500
   d tvrh2q(tv(lev=850),rh(lev=850))

   In this case, the **tvrh2q** function would assume a pressure of 500mb, which is the current dimension environment setting for the Z dimension. However, we are providing data from the 850mb level, so the function will produce incorrect results.

tvrh2t

tvrh2t(tvexpr,rhexpr)

Given virtual temperature and relative humidity, **tvrh2t** returns the temperature in degrees Kelvin. The operation of this function is the same as **tvrh2q**; refer to the above description for more information.

Special Purpose Functions

tloop

tloop(*expr*)

When time is a varying dimension in the dimension environment, the **tloop** function evaluates the ***expr*** at fixed times, then reconstructs the time series to obtain a final result that is time varying. The **tloop** function is required due to the implementation of the GrADS expression evaluation rules, and the implementation of certain other functions. The **tloop** function can also improve performance for certain calculations.

The **tloop** function is provided as a way to obtain time series from functions that themselves are not implemented to be able to operate when time is a varying dimension. See the examples below.

Usage Notes

1) The **tloop** function loops through time based on the time increment of the default file; it is thus important to have the default file set appropriately.

2) The **tloop** function and the **define** command work very similarly. In many cases, the **define** command can be used to obtain the same result as using **tloop**. In fact, the **define** command can be even more useful along those lines, since it also loops through the Z dimension, in effect creating a **zloop** function. See the **define** command for more information.

Examples

1) A typical application of the **tloop** function is to calculate a time series of areal averages using the **aave** function. Since the **aave** function will not work when time is a varying dimension, the use of **tloop** is required:

   set x 1
   set y 1
   set t 1 31
   d tloop(aave(ts,lon=0,lon=360,lat=-90,lat=90))

   Note that the dimension environment is set up to reflect the kind of plot desired, namely a **line** plot where time is the varying dimension. Thus it is necessary to fix the X and Y dimensions; the values of those dimensions in this case are not relevant. Using define can achieve the same effect, i.e.,

   define t=aave(ts,lon=0,lon=360,lat=-90,lat=90)

   d t

2) The **tloop** function can be used to smooth in time:

   set lon -180 0
   set lat 40
   set lev 500
   set t 3 28
   d tloop(ave(z,t-2,t+2))

   In this example, we are plotting a time-longitude cross section, where each time is a 5 time period mean centred at that time.

3) If we wanted to display a time-longitude cross section (X and T varying), with the data being averaged over latitude, the 'standard' way to do this might be:

   set lon -180 0
   set lat 40
   set lev 500
   set t 1 31
   d ave(z,lat=20,lat=40)

   This calculation could be fairly time consuming, since to perform the average, a longitude-time section is obtained at each latitude. If the time period is long, then this would be a very inefficient operation, due to the ordering of data in a typical GrADS data set. The **tloop** function might substantially improve the performance of this calculation:

   d tloop(ave(z,lat=20,lat=40))

   since the average is then done at each fixed time, and is thus just an average of X varying data over Y. Thus the **tloop** function here is simply being used to force a different ordering to the calculation, although the result is the same.

Station Data Functions

gr2stn

gr2stn(grid_expr,stn_expr)

Performs an interpolation from grid space back to station locations, where:

grid_expr is any valid GrADS expression that gives a grid result. The interpolation will be done using this gridded data.
stn_expr is any valid GrADS expression that gives a station data result. The interpolation will be done *to these station locations*, the value of the station data is not used.

The result of the function is station data. The interpolation is done bi-linearly within the grid space. No weighting is done to account for the world coordinate systems.

Examples

1) To examine the difference between an analysis (i.e., gridded data) and the original observations, one could:

    d t.3-gr2stn(t.1,t.3)

    where file 1 is gridded data, and file 3 is station data. The result would display as differences at the station locations.

2) If one wanted to display the difference calculated in Example 1 as a contour field, one can use the **oacres** function to do a quick analysis of the station values:

    d oacres(t.1,t.3-gr2stn(t.1,t.3))

oacres

oacres(grid_expr,stn_expr<,radii<first guess>>)

A Cressman objective analysis is performed on the station data to yield a gridded result representing the station data:

grid_expr An expression that has a gridded data result. The actual values of this grid are ignored; the grid is used as a template to perform the analysis. The scaling of this grid must be linear in lat-lon.
stn_expr An expression that has a station data result. The station data is analyzed to the grid.
radii Optional radii of influence. Multiple radii are usually provided, separated by commas. If not provided, default radii are used, in grid units: 10,7,4,2,1. The third radius specified is special, in that any grid points that do not have stations within that radius are set to the missing data value. See below for a discussion of the radii of influence.

The Cressman Analysis scheme is described in a paper in Monthly Weather Review, 1959. In summary, multiple passes are made through the grid at subsequently lower radii of influence. At each pass, a new value is determined for each grid point by arriving at a correction factor for that grid point. This correction factor is determined by looking at each station within the radius of influence from the grid point. For each such station, an error is determined as the difference between the station value and a value arrived by interpolation from the grid to that station. A distance weighted formula is then applied to all such errors within the radius of influence of the grid point to arrive at a correction value for that grid point. The correction factors are applied to all grid points before the next pass is made.

Usage Notes

1) The **oacres** function can be quite slow to execute, depending on grid and station data density.
2) The Cressman Analysis scheme can be unstable if the grid density is substantially higher than the station data density (i.e., far more grid points than station data points). In such cases, the analysis can produce extrema in the grid values that are not realistic. It is thus suggested that you examine the results of **oacres** and compare them to the station data to ensure they meet your needs.
3) In general, objective analysis is a complex topic, and many schemes for doing it have been developed over the years. The **oacres** function is provided more as a quick-look feature rather than a rigorous analysis scheme. If you have specific analysis requirements, consider doing your objective analysis outside of GrADS with special purpose programs.

Examples

1) In the simplest case:

   oacres(ts,ts.2)

2) To specify your own radii of influence:

   oacres(ts,ts.2,12,8,5,4,3,2,1)

3) Setting the first guess in oacres

   oacres sets the initial value of the analysis grid to the arithmetic average of the obs in the area. For positive definite quantities like precipitation, this can produce an unrealistic analysis in regions of **no obs** (e.g., no rain is better guess than average rain).

   The call to **oacres** (stands for Cressman r**2 scan analysis) has the form,

   d oacres(grid,obs,r1,r2,r3,...,r30)

   where:

grid	is a grid to which the obs will be analyzed to
obs	is the obs "grid"
r1,...,r30	are the scan radii in GRID UNITS.

   The default scan radii are:

   r1=10.0
   r2=7.0
   r3=4.0
   r4=2.0
   r5=1.0

   or five radii. This is good for meteorological fields, but may not yield a desirable analysis for hydrologic fields which are not as continuous. The number of radii can be changed, up to a maximum of 30, to accommodate the requirements of different types of station data.

   To change the first guess, set the penultimate **r** to **-1** and the last **r** to the desired first guess. For example,

   'd oacres(pr.1,pr.2,5,4,-1,-0.01)'

would do an analysis of the **pr.2 obs** to the **pr.1 grid** with **2 scan radii** of **5** and **4** grid units with a first guess of **-0.01**. A first guess of **0** can be used to reduce the tendency of **oacres** to create artificial *bullseyes* for spatially discontinuous fields such as precipitation.

Errors in setting up the first guess produce the default oacres.

stnave

stnave(expr,dexpr1,dexpr2<,-m cnt>)

Takes an average of station data over time:

expr A valid GrADS expression that gives a station data result.
dexpr1 A dimension expression giving the starting time for the average.
dexpr2 A dimension expression giving the ending time for the average.
m *cnt* Optional minimal data count for the average to be taken. If, in the time series, there are fewer available data points for a particular station than the ***cnt*** value, then the result for that station is the missing data value. The default ***cnt*** value is 1 (ie, even 1 valid station in a time series of even thousands of points would give a valid result for that station).

Usage Notes

1) The times are looped through based on the time interval of the default file. It is thus very important to set the default file to that of the station data file, or a file with the same time interval, or not all station reports will be included in the average.
2) If there is more than one report per station for a particular time, those reports are averaged equally to arrive at a single value for that time. The final average consists of each report for each time being averaged, with missing times not included in the average.
3) Reports from different times are considered to be for the same station when the station id, the latitude, and the longitude all match exactly.

Examples

1) A typical usage of the **stnave** function would be:

   stnave(ts,t=1,t=20,-m 10)

   Here an average is taken over 20 times, and if there are fewer than 10 reports for a station then that station will be missing in the final result.

stnmin

stnmin(expr,dexpr1,dexpr2<,-m cnt)

Examines a time series of station data and returns the minimum value encountered for each station. Operands and usage are the same as the **stnave** function; see above.

stnmax

stnmax(expr,dexpr1,dexpr2<,-m cnt)

Examines a time series of station data and returns the maximum value encountered for each station. Operands and usage are the same as the **stnave** function; see above.

Vector Functions

hcurl

hcurl(uexpr,vexpr)

Calculates the vertical component of the curl (i.e., vorticity) at each grid point using finite differencing on the grids provided. It is assumed that ***uexpr*** gives the U Wind component, and that ***vexpr*** provides the V Wind component.

Usage Notes

1) The algorithm used for the finite difference calculation is described as an Example for the **cdiff** function.
2) The function assumes an X-Y varying dimension environment, and will not operate unless that is the case. The **define** command can be used in conjunction with the **hcurl** function to create 3 or 4 dimensional fields of vorticity, from which vertical cross-sections could be displayed.
3) The boundaries of the grid are set to missing.
4) The radius of the earth used in the calculation is in meters; thus the units of the wind expressions provided would normally be m/s.

Examples

1) To display the vorticity:

   d hcurl(u,v)

2) If you want to display a vertical cross section of vorticity, you first need to calculate it over a 3-Dimensional region:

   set lon 0 360
   set lat -90 90
   set lev 1000 100
   define vort = hcurl(u,v)
   set lon -90
   display vort

hdivg

hdivg(uexpr,vexpr)

Calculates the horizontal divergence using finite differencing. Exactly the same as **hcurl** in all other respects; see the Usage Notes and Examples above.

Usage Notes

1) The numerical stability of calculating horizontal divergence using finite differencing is very low. Please use the function with caution.

mag

mag(uexpr,vexpr)

Performs the calculation: sqrt(**uexpr*uexpr**+**vexpr*vexpr**). May be used with gridded or station data.

21.0 User Defined Functions (UDFs):

Users may write their own GrADS functions in the computer language of their choice, and have them available from the GrADS expression facility (via the **display** command). Some possible user defined functions might be:

- filtering functions
- grid interpolation functions
- thermodynamic functions

You may write a function that can be invoked via the GrADs expression facility. This function may be written in any computer language, and may perform any desired I/O, calculations, etc. ***You should read the following documentation carefully to understand the restrictions to this capability.***

Overview of User Defined Functions

The steps that GrADS uses to invoke a user defined function are:

1) When GrADS is first started, it reads a file that describes the user defined functions. This file is called the 'user defined function table'.
2) When a user function is invoked via the display command expression, GrADS parses the arguments to the functions, obtains the results of any expressions, and writes the resultant data to a 'function data transfer file'.

   Please note that In a user-defined function adding the double quote (**“”**) around a **char** argument passes the string directly *without* the usual conversion to lower case and removal of blanks, e.g.,

   d grhilo(slp,F8.2,"This is the Label",0.25)

   Here **F8.2** is passed as **f8.2**, but the second character string would not be converted to **thisisthelabel**.
3) A user written program is then invoked. This program may read the function data transfer file, do any desired processing, then write the result into a function result file.
4) GrADS will read the function result file and generate the internal objects necessary for this result to participate in the remainder of the expression evaluation.

The user defined function table

The user defined function table is a flat text file that contains information about each user defined function. There are five records for each defined function, and the file may contains descriptions for any number of functions.

The 5 records are:

Record 1: This record contains several blank delimited fields:

Field 1: The name of the function, 1-8 characters, beginning with a letter. The name should be in lower case. Note that function names are not case dependent, and that GrADS converts all expression to lower case before evaluation.
Field 2: An integer value, specifying the minimum number of arguments that the function may have.

Field 3: An integer value, specifying the maximum number of arguments that the function may have. This may not be more than 8.

Field 4 to N: A keyword describing the data type of each argument:

expr: The argument is an expression.
value: The argument is a data value.
char: The argument is a character string.

Record 2: This record contains several blank delimited option keywords. Current options:

sequential GrADS will write data to the function data transfer file in FORTRAN sequential unformatted records.

direct GrADS will write data to the function data transfer file without any record descriptor words.

Note: **sequential** is typically appropriate if the function routine is written in FORTRAN. **direct** is more appropriate for C.

Record 3: This record contains the file name of the function executable routine. This routine will be invoked as its own separate process via the **'system'** call. Do a **'man system'** if you would like more information on the rules governing this system feature.

Record 4: This record contains the file name of the function data transfer file. This is the file that GrADS will write data to before invoking the user function executable, and is typically the file the function will read to obtain the data to be operated upon.

Record 5: This record contains the file name of the function result file. The function writes the result of its operations into this file in a specified format, and GrADS reads this file to obtain the result of the function calculation.

The user function definition table itself is pointed to by the environment variable GAUDFT. If this variable is not set, the function table will not be read. An example of setting this variable is:

setenv GAUDFT /usr/local/grads/udft

User defined functions have precedence over GrADS intrinsic functions, thus a user defined function can be set up to replace a GrADS function. Be sure you do not do this inadvertently by choosing a function name already in use by GrADS.

Format of the function data transfer file

The function data transfer file contains a single header record, then contains one or more records representing each argument to the function. The user function routine will know what data types to expect (since they will be specified in the UDFT), and can read the file in a predictable way. The file format may seem somewhat complex at first, but later examples will show that it is not as bad as it may seem at first glance.

Header record: The header record always contains 20 floating point numbers. The record will always be the same size. Values defined in this record are:

1st value: Number of args user used when invoking the function
2nd value: Set to zero, to indicate this particular transfer file format. The function should test this value, and return an error if non-zero, in order to be compatible with future enhancements to this file format.
Values 2 to 20: Reserved for future use.

Arg records: Each argument type will result in a specific set of records being written out. The records are written in the order that the arguments are presented. For each data type:

value: A record will be written containing a single floating point value

char: A record will be written containing the character value of the particular argument. The length of the record will be 80 bytes. If the argument is longer, the trailing bytes will be lost. If the argument is shorter, it will be padded with blanks. Note that the argument will already be processed by the GrADS expression parser to some extent, which will convert all characters to lower case and remove any blanks.

expr: When the argument is an expression, GrADS will evaluate the expression and write the result to the transfer file. Currently only gridded data is supported. Several records will be written to the file for each expr type argument:

1st record: The grid header record. This record contains 20 values, all floating point. Note that some of the values are essentially integer, but for convenience they are written as a floating point array. Appropriate care should be taken in converting these values back to integer.

1: Undefined value for the grid
2: i dimension (idim). Dimensions are:
  1 - None
  0 - X dimension (lon)
  1 - Y dimension (lat)
  2 - Z dimension (lev)
  3 - T dimension (time)
3: j dimension (jdim). Note: if idim and jdim are -1, the grid is a single value. If jdim is -1, the grid is a 1-D grid.
4: number of elements in the i direction (isiz)
5: number of elements in the j direction (jsiz)
  Array is dimensioned (isiz,jsiz).
6: i direction linear flag. If 0, the dimension has non-linear scaling.
7: j dimension linear flag.
8: istrt. This is the world coordinate value of the first i dimension, ONLY if the I dimension has linear scaling and the i dimension is not time.
9: iincr. Increment in the i dimension of the world coordinate. ONLY if the i dimension has linear scaling.
10: jstrt. World coordinate of the first j dimension, only if the j dimension has linear scaling, and the j dimension is not time.
11: jincr. World coordinate increment for j dimension.
12: If one of the dimensions is time, values 12 to 16 are defined as the start time:
  12 is the start year.
  13: start month
  14: start day
  15: start hour
  16: start minute
17: Values 17 and 18 contain the time increment for the time dimension.
  17 contains the increment in minutes.
18: increment in months. (GrADS handles all increments in terms of minutes and months).
19,20: reserved

2nd record: This contains the grid data. It is isiz*jsiz number of floating point elements.

Possible 3rd record. If the i or j dimension scaling is non-linear, the world coordinate values at each integral i(j) dimension value is written. Thus, if the i dimension is non-linear, isiz number of elements will be written. If the j dimension is non-linear (and the i dimension IS linear), then jsiz elements will be written.

Possible 4th record. Only written if both the i and j dimension have non-linear scaling. In this case, this record contains the j dimension world coordinate values; jsiz number of floating point elements.

The existence of the 3rd or 4th records can only be determined by examining the grid header record contents. Note that the time dimension is *always* linear as currently implemented in GrADS.

Note that it is not necessary for the function to handle all possible perturbations of argument data. The function may test for certain conditions and return an error code if those conditions are not met.

Format of the function result file

The function result file returns the result of a function to GrADS. It is the responsibility of the function process to write this file in the proper format. ***A file written out in an improper format may cause GrADS to crash, or to produce incorrect results.***

The result of a function is always a grid. Thus, the format of the function result file is:

First, a header record.

This record contains 20 floating point values. The first value contains the return code. Any non-zero value causes GrADS to assume the function detected an error, and GrADS does not read any further for output. Value 2 should currently always be set to zero (indicating a simple result file), and values 3 to 20 are reserved for future use.

Next, a set of grid records

These records are in exactly the same order and format as in the data transfer file. (Note that if the function is returning a grid that has the same scaling and dimension environment as an argument grid, the records for that argument grid may be written out to the result file unmodified—only the data need change).

Example: Linear Regression Function

This is a simple example of what a user defined function might look like in FORTRAN. This is a simple linear regression function, which only handles a 1-D grid and takes one argument, and expression.

First, the function definition table:

linreg 1 1 expr
sequential
/mnt/grads/linreg
/mnt/grads/linreg.out
/mnt/grads/linreg.in

The FORTRAN program is compiled, and named linreg, and placed in /mnt/grads. The program source code is:

```
      real vals(20),ovals(20)
      real x(10000),y(10000)
c
      open (8,file='/mnt/grads/linreg.out',form='unformatted')
      open (10,file='/mnt/grads/linreg.in',form='unformatted')
c
      read (8)
      read (8) vals
      idim = vals(2)
      jdim = vals(3)
c
c  If this is not a 1-D grid, write error message and exit
c
      if (idim.eq.-1 .or. jdim.ne.-1) then
        write (6,*) 'Error from linreg: Invalid dimension environment'
        vals(1) = 1
        write (10) vals
        stop
      endif
c
c  If the grid is too big, write error message and exit
c
      isiz = vals(4)
      if (isiz.gt.10000) then
        write (6,*) 'Error from linreg: Grid too big'
        vals(1) = 1
        write (10) vals
        stop
      endif
c
c  Read the data
c
      read (8) (y(i),i=1,isiz)
c
c  Read non-linear scaling if necessary
c
      ilin = vals(6)
      if (ilin.eq.0) then
        read (8) (x(i),i=1,isiz)
      else
        do 100 i=1,isiz
          x(i) = i
 100    continue
      endif
c
c  Do linear regression
c
      call fit (x,y,isiz,a,b)
c
c  Fill in data values
c
      do 110 i=1,isiz
        y(i) = a+x(i)*b
 110  continue
c
c  write out return info.  The header and the non-linear scaling
c  info will be the same as what GrADs gave us.
c
```

```
      ovals(1) = 0.0
      write (10) ovals
      write (10) vals
      write (10) (y(i),i=1,isiz)
      if (ilin.eq.0) write(10) (x(i),i=1,isiz)
c
      stop
      end
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      SUBROUTINE FIT(X,Y,NDATA,A,B)
C
C  A is the intercept
C  B is the slope
C
      REAL X(NDATA), Y(NDATA)
C
      SX = 0.
      SY = 0.
      ST2 = 0.
      B = 0.
      DO 12 I = 1, NDATA
        SX = SX + X(I)
        SY = SY + Y(I)
 12   CONTINUE
      SS = FLOAT(NDATA)
      SXOSS = SX/SS

      DO 14 I = 1, NDATA
        T = X(I) - SXOSS
        ST2 = ST2 + T * T
        B = B + T * Y(I)
 14   CONTINUE
      B = B/ST2
      A = (SY - SX * B)/SS
      RETURN
      END
```

22.0 Further Features of GrADS Data Sets

This chapter provides more information on controlling and specifying GrADS data sets. Whereas the material in Chapter 4 is fundamental to running GrADS, the information presented here gives enhanced flexibility and control of data sets which many users will not immediately require A full understanding of the contents of Chapter 4 **Using GrADS Data Files** is assumed.

File and time group headers

You may tell GrADS that your data file has a header. To do this, include the record in your descriptor file:

fileheader length

where **length** is the number of bytes in the header. GrADS will skip past this header, then treat the file as though it were a normal GrADS file after that point. This option is valid only for GrADS gridded data sets.

Variable format/structure control

This feature allows control of the structure and format of each variable.

In a GrADS data descriptor file each variable is defined using the following syntax,

vname nlevs units1,units2,unit3,units4 vdesc where:

vname = the name as it is referenced in GrADS
nlevs = the number of levels in the vertical or z dimension
units? = a sequence of one to four ints used to define the variable for GRIB processing and for specialized handling.

These features are invoked through the **units?** parameters according to the following syntax:

-1,xx,yy where:

xx = structure
yy = additional attributes
-1 is used to tell GrADS special formatting is happening.

There are four structures supported......

1) xx = 10

Data where the variable and levels are transposed, (**lon,lat,var,lev,time** instead of **lon,lat,lev,var,time**). For example, suppose you have four variables,

u(x,y,z),v(x,y,z),t(x,y,z),z(x,y,z)

and you want to write them out in so they can be viewed in GrADS.

In FORTRAN you would have,

```
parameter (ni=144,nj=91,nk=18,nt=4)
dimension u(ni,nj,nk),v(ni,nj,nk),t(ni,nj,nk),z(ni,nj,nk),dum(ni,nj)
do n=1,nk
  call load(u,ni,nj,nk,n,dum)
```

```
    write(10) dum
  end do
  do n=1,nk
    call load(v,ni,nj,nk,n,dum)
    write(10) dum
  end do
  do n=1,nk
    call load(t,ni,nj,nk,n,dum)
    write(10) dum
  end do
  do n=1,nk
    call load(z,ni,nj,nk,n,dum)
    write(10) dum
  end do
  subroutine load(a,ni,nj,nk,n,dum)
  dimension a(ni,nj,nk),dum(ni,nj)
  do i=1,ni
    do j=1,nj
    dum(i,j)=a(i,j,n)
  end do
  end do
  return
```

and the **.ctl** would look something like:

```
dset ^model.dat
title some model data
undef   0.10000E+16
options sequential
xdef 144 linear   0 2.5
ydef  91 linear -90 2.0
zdef  18 levels
1000.000  950.000  900.000  850.000  800.000  700.000  600.000  500.000
  400.000  300.000  250.000  200.000  150.000  100.000   70.000   50.000
    30.000   20.000
tdef   4   linear apr85 1mo
vars 4
    u 18 0 u component from NASA model
    v 18 0 v component from NASA model
    t 18 0 temperature from NASA model
    z 18 0 geopotential height from NASA model
endvars
```

However, in NASA GCM “phoenix” format, *for the upper air prog variables only,* they have,

```
  do n=1,nk
    call load(u,ni,nj,nk,n,dum)
    write(10) dum
    call load(v,ni,nj,nk,n,dum)
    write(10) dum
    call load(t,ni,nj,nk,n,dum)
    write(10) dum
    call load(z,ni,nj,nk,n,dum)
    write(10) dum
  end do
```

Thus, variables and z are transposed or all variables are written out one level at a time....

To make matters trickier, for the **upper air diagnostics**, the **NASA** format reverts the GrADS convention, so now we need to tell GrADS that the **var-z** transposed is no longer active...

To handle the upper air prog variables, and then upper air diagnostics (e.g., cuheat and clouds), in the **.ctl** file we would have:

```
dset ^model.nasa.dat
title some model data from NASA
undef   0.10000E+16
options sequential
xdef 144 linear   0 2.5
ydef  91 linear -90 2.0
zdef  18 levels
1000.000  950.000  900.000  850.000  800.000  700.000  600.000  500.000
  400.000  300.000  250.000  200.000  150.000  100.000   70.000   50.000
    30.000   20.000
tdef   4   linear apr85 1mo
vars 6
    u      18 -1,10,1 u component from NASA model
    v      18 -1,10,1 v component from NASA model
    t      18 -1,10,1 temperature from NASA model
    z      18 -1,10,1 geopotential height from NASA model
    cuheat 18 -1,10,2 cumulus heating
    clouds 18 -1,10,2 cloud fraction
endvars
```

Thus,

yy = 1 means the variables have been **var-z transposed**
yy = 2 means the variables are now "**normal**"

2) xx = 20

This handles 4-D variables, i.e., all times for one variable written out in one chunk as opposed to writing all variables at one time and then all variables at the next time. From the previous example, let's assume you now have data at **nt** times...

In a typical model you would have,

```
    parameter (ni=144,nj=91,nk=18)
    dimension u(ni,nj,nk),v(ni,nj,nk),t(ni,nj,nk),z(ni,nj,nk),dum(ni,nj)
    do l=1,nt
C run model and update the prog variables
        call prog(u,v,t,z)
        do n=1,nk
            call load(u,ni,nj,nk,n,dum)
            write(10) dum
        end do
        do n=1,nk
            call load(v,ni,nj,nk,n,dum)
            write(10) dum
        end do
        do n=1,nk
            call load(t,ni,nj,nk,n,dum)
            write(10) dum
        end do
        do n=1,nk
```

```
        call load(z,ni,nj,nk,n,dum)
        write(10) dum
      end do
    end do
```

And you'd have no problem reading the data in GrADS, but suppose you now read the model output and write out the u,v and t data differently,

```
    parameter (ni=144,nj=91,nk=18,nt=4)
    dimension u(ni,nj,nk),v(ni,nj,nk),t(ni,nj,nk),z(ni,nj,nk),dum(ni,nj)
    open (10...)
    open (12...)

    do l=1,nt
C write out all the u's
      do n=1,nk
        read(10) dum
        write(12) dum
      end do
      do n=1,nk
        read(10) dum
      end do
      do n=1,nk
        read(10) dum
      end do
      do n=1,nk
        read(10) dum
      end do
    end do
    rewind 10

C  now write out all the v
    do l=1,nt
      do n=1,nk
        read(10) dum
      end do
      do n=1,nk
        read(10) dum
        write(12) dum
      end do
      do n=1,nk
        read(10) dum
      end do
      do n=1,nk
        read(10) dum
      end do
    end do
    rewind 10

C  now write out all the t
    do l=1,nt
      do n=1,nk
        read(10) dum
      end do
      do n=1,nk
        read(10) dum
      end do
      do n=1,nk
        read(10) dum
        write(12) dum
      end do
```

```
    do n=1,nk
      read(10) dum
    end do
  end do
```

While this seems unnatural for a model, some data sets look like this with several variables containing all times.

The GrADS **.ctl** file for the above example, would use:

```
undef   0.10000E+16
options sequential
xdef 144 linear   0 2.5
ydef  91 linear -90 2.0
zdef  18 levels
1000.000  950.000  900.000  850.000  800.000  700.000  600.000  500.000
  400.000  300.000  250.000  200.000  150.000  100.000   70.000   50.000
   30.000   20.000
tdef   4   linear apr85 1mo
vars 3
    u  18 -1,20 u component from NASA model
    v  18 -1,20 v component from NASA model
    t  18 -1,20 temperature from NASA model
endvars
```

The **sequential** option is set because we wrote the data using unformatted (f77) I/O.

Now suppose you want to use the template option in time. Use **yy** to tell GrADS how many times there are in each file, e.g.,

```
dset ^mydate.%y2.dat
options sequential template

.
.
.
tdef 120 linear jan79 1mo
vars 3
    u 18 -1,20,12 u component
    v 18 -1,20,12 v component
    t 18 -1,20,12 temperature all
endvars
```

yy=12 tells GrADS there are 12 months in each file.

3) xx = 30

This handles a pathological case were lon and lat are transposed or you have (lat,lon) as opposed to (lon,lat) data. While this does "work" it is **very inefficient** because we didn’t want to make a big change to GrADS internal I/O to handle this unusual case.

However, it is useful for initial inspection and debugging and that’s only what it is designed for. Here’s an example **.ctl** file

```
dset ^latlon.dat
title test case of data lat and lon are transposed a(j,i) vice a(i,j)
```

```
undef 1e20
xdef 144 linear 0 2.5
ydef  73 linear -90 2.5
zdef   1 levels 1013
tdef   1 linear 00z1jan1995 12hr
vars 1
    u 0 -1,30 u comp
endvars
```

4) **xx = 40**

This option handles **non float data** by internal conversion to floats after the read.

There are two suboptions (**yy**)

yy=1 -one-byte unsigned ints (0-255)
yy=4 -integer data (4 byte on 32-bit machines and 8-byte on crays)

The first case was to handle GMS data on a CD-ROM from MRI in Tsukuba, Japan. Here is the gms **.ctl** file:

```
dset ^I921110.Z12
undef 1e20
title GMS IR imagery during TOGA COARE
fileheader 500
options yrev
xdef 500 linear 130.05 0.1
ydef 300 linear -14.95 0.1
zdef   1 levels 1013
tdef   1 linear 00Z1nov1992 12hr
vars 1
    tb 0 -1,40,1 IR brightness temp - 100 K
endvars
```

The **yy=4** option has been used for integer data representing surface type...

Multiple file time series

GrADS now allows you to handle many actual data files as one GrADS file, if the individual data files are in a GrADS readable format, and if the files are split along time. In the initial implementation, the time(s) that are in each file are indicated by the file name.

An example of this might be hourly data, where each 24 hours has been placed in a separate file. Each file is named this way:

```
1may92.dat
2may92.dat
etc.
```

You indicate to GrADS that there are multiple files in this time series by giving a substitution template as the file name:

```
dset %d1%mc%y2.dat
```

and giving an **options** record that looks like:

options template

and specifying the time range and increment in the **tdef** record:

tdef 72 linear 0z1may1993 1hr

GrADS will figure out automatically that there are 24 times in each file, and what file names correspond to what times. As you display data, GrADS will only open one file at a time. As you change times such that another file is referred to, the open file is closed, and the new file is opened.

Valid substitutions are:

%y2	- 2 digit year (last 2 digits)
%y4	- 4 digit year
%m1	- 1 or 2 digit month
%m2	- 2 digit month (leading zero if needed)
%mc	- 3 character month abbreviation
%d1	- 1 or 2 digit day
%d2	- 2 digit day
%h1	- 1 or 2 digit hour
%h2	- 2 digit hour
%h3	- 3 digit hour (e.g., 120 or 012)
%f2	- 2 or 3 digit forecast hour
%f3	- 3 digit forecast hour
%n2	- 2 digit minute (leading zero if needed)

for specifying the initial time (e.g., NWP model output from NMC and FNMOC)

%iy2	- initial 2 digit year (last 2 digits)
%iy4	- initial 4 digit year
%im1	- initial 1 or 2 digit month
%im2	- initial 2 digit month (leading zero if needed)
%in2	- initial 2 minute (leading zero if needed)
%imc	- initial 3 character month abbreviation
%id1	- initial 1 or 2 digit day
%id2	- initial 2 digit day
%ih1	- initial 1 or 2 digit hour
%ih2	- initial 2 digit hour
%ih3	- initial 3 digit hour

(time increment must be hours)

This support works on all supported GrADS data types (GrADS gridded, GRIB, GrADS station data). If you specify file format options, the options must apply equally to each file.

The real-time data on DECstations makes use of this new feature. See the data descriptor files:

/data/wx/grads/sa.ctl
/data/wx/grads/sareps.ctl
/data/wx/grads/wx.ctl

for additional examples.

Enhanced data formats and structures

The GrADS I/O layer has been modified to handle extra data structures and types. This was initially prompted by a need to work with the **NASA GSFC DAO** reanalysis and **GCM** output data in its own "**phoenix**" format and the **Climate Analysis Center's Climate Diagnostic Data Base (CDDB)**. These options define **global** characteristics of the data file......

In the data descriptor file the follow keywords have been added,

theader ttttt
xyheader xxxxx where:

ttttt = the number of header bytes preceding the actual data for each time block (e.g., each 4-D lon,lat,lev,var in time)
xxxxx = the number of header bytes which precede the data for each xy block (e.g., a 2-D lon/lat field).

N.B. Using these features requires a *detailed* understanding of your data! GrADS will read the data file *exactly* the way you tell it to! Mistakes here will wreck your results.

A FORTRAN program to automatically build the **.ctl** file from the **NASA phoenix** format is available from fiorino@typhoon.llnl.gov.

23.0 Programming GrADS: Using the Scripting Language

The GrADS scripting language, used via the GrADS **run** command, provides a similar capability to the **exec** command, except that a script may have variables, flow control, and access GrADS command output. Scripts may be written to perform a variety of functions, such as allowing a user to point and click on the screen to select something, to animate any desired quantities, to annotate plots with information obtained from GrADS **query** commands.

Overview of the Scripting Language

The scripting language is similar to REXX in implementation. All variables are of type STRING. Operations are supported on script variables. Flow control is achieved via if/else/endif and while/endwhile constructs. Loop flow may be modified by the continue or break commands. Strings contained in variables or generated via an expression may be issued to GrADS as commands. The result of those commands (the string that GrADS would have typed on the terminal) is put into a variable and made available to the script. The language includes support for functions.

Elements of the Language

A script file is split into records. The end of a script record is determined by either a newline character (end of record for the file) or a semicolon (where the semicolon is not contained within a constant string).

Each script record may be one of the following script record types:

- Assignment
- If / Else / Endif
- while / endwhile / break / continue
- function header / return
- say / pull
- comment

If a script record is none of the above, it is assumed to be an statement record, which contains a script expression. The result of the expression is passed to GrADS as a command for execution. The text result of the GrADS command is put in the variable **’result’** for examination by the script.

Many of the above record types will contain expressions. Script expression are composed of operators and operands, where the operands are script variables, function calls, or constants, and the operators are mathematical, logical, or concatenation operations.

There is no **’goto’** in this language.

N.B. GrADS needs a **carriage return** after the last command line in the script file, otherwise GrADS won’t execute this command line.

Variables

Script Language variable names are 1 to 8 characters, beginning with an alphabetic character and containing letters or numbers only. The name is case sensitive.

The contents of a script variable is always a character string. For some operations, the character string will be interpreted as a number.

If a variable has not yet been assigned, its value is its name.

If the contents of a variable or string constant are a number in the correct format, certain operators will perform numeric operations, giving a string result which will also be a number.

String variables

String variables or string constants are enclosed in either single **‘’** or double **“”** quotes. For example:

name = ‘Peter Pan’
name = “Peter Pan”

Predefined variables

Some variable names are **predefined**, it is a good idea to avoid assigning values to these variables. The following are predefined variables:

lat
lon
lev
result
rec

Global scripting variables

Scripting variables are usually local to the functions they are contained in. Global scripting variables are also available. They are specified via the **variable name**. Any variable name starting with an underscore (_) will be assumed to be a global variable, and will keep its value throughout an entire script file. For example:

_var1 = “global variable 1”

N.B. global variables *cannot* be used in function headers: Thus:

function dostuff(_var)

wouldn’t make sense, since **_var** is a global variable, and would be invalid if it were the *only* argument.

Compound scripting variables

The scripting language supports compound variables, which can be used to construct arrays in scripts. A compound variable has a variable name with segments separated by periods. For example:

varname.i.j

In this case, when the **variable** contents are accessed, **i** and **j** will be looked up to see if they are also variables (non-compound). If they are, the **i** and **j** will be replaced by the string values of **i** and **j**.

For example:

```
i = 10
j = 3
varname.i.j = 343
```

In the above example, the assignment is equivalent to:

```
varname.10.3 = 343
```

Note that the string values of **i** and **j** may be anything, but the **variable** name specification in the script must follow the rules for variable names: letters or numbers, with a leading letter. The variable name after substitution may be any string:

```
i = 'a#$xx'
varname.i = 343
```

The above is valid. However, we cannot refer to this variable name directly:

```
varname.a#$xx = 343
```

would be invalid.

Variable names may *not* be longer than 16 characters, either *before* or *after* substitution.

Note that the GrADS scripting language is not particularly efficient in handling large numbers of variables. Thus compound variables should not be used to create large arrays:

```
i = 1
while (i<10000)
   var.i = i
endwhile
```

The above loop will create 10000 distinct variable names. Having that number of variables in the variable chain will slow the script down a lot. If this turns out to be a poor design choice, let me know and I will consider making the variable handling more efficient.

Operators

The following operators are implemented:

- **|** logical OR
- **&** logical AND
- **!** unary NOT
- **-** unary minus
- **=** equality
- **!=** not equal
- \> greater than
- \>= greater than or equal
- < less than
- **<=** less than or equal
- **%** concatenation
- **+** addition
- **-** subtraction
- ***** multiplication
- **/** division

The following operators will perform a numeric operation if the operands are numeric:

=, !=, >, >=, <, <=, +, -, *, /

If any of the following operations are attempted with non-numeric operands, an error will result:

+, -, *, /

Arithmetic operations are done in floating point. If the result is integral, the result string will be an integer.

A **logical** operator will give a character **0** (zero) if the result is FALSE, and a character **1** (one) if the result is TRUE.

Expressions

Script expressions consist of operands, operators, and parentheses.

The precedence of operators is:

-, ! (Unary)
/, *
+, -
%
=, !=, >, >=, <, <=
&
|

Within the same precedence level, operations are performed left to right. Parentheses modify the order of operation according to standard convention.

Operands may be variables (discussed earlier), string constants, or function calls. String constants are enclosed in either single or double quotes. Numeric constants may be entered without quotes, but are still considered string constants.

An example of a string constant:

’This is a string’

The entire expression, including all function calls, etc. will be performed to obtain a result. For example:

var1!=’’ & var1*var2<10

In this expression, both sides of the logical AND operation will be resolved, and the subexpression to the right might result in an error. In these cases, a double nested if will be required.

In some cases, the concatenation operator is implied. This takes place whenever two operands abut (with or without intervening blanks—the blanks are ignored).

For example, the following expressions have the same effect:

var1%var2%’String’	uses the concatenation operator **%**
var1 var2’String’	implied concatenation

giving:

‘var1var2String’

Keep in mind the order of precedence for the concatenation operator.

Function calls take the form of:

name(arg,arg,arg,...)

where the name follows the same rules as for variable names, and the args may be expressions.

Flow control

IF Blocks

Flow of control may be controlled via the **if/else/endif** construct. The format is:

```
if expression          Must be on separate record
   script record
   script record
   .
   .
else                   Optional
   script record
   .
   .
endif                  Required!
```

Note that the following script record is invalid:

if (i=10) j=20

You would instead need to enter three script records:

```
if (i=10)
   j = 20
endif
```

You could enter these three script records on the same line:

if (i=10); j=20; endif;

The portion of the if block executed depends on the result of the expression. If the expression resolves to a string containing the character 0, the **'else'** portion is executed. If the result string is *anything* else, the **'if'** portion is executed.

N.B. There is no **ELSE IF** construct in GrADS.

WHILE Blocks

The **while** construct is as follows:

```
while expression       On separate script record
   script record
   script record
   .
   .
endwhile               Required!
```

While the expression is true—ie, is not exactly equal to a character 0 -- the loop is executed.

Two additional script commands can be used to modify the loop execution. **break** will end execution of the loop immediately. **continue** will branch immediately back to the top of the loop, and the expression will be re-evaluated.

For example:

```
t = 1
while (t<10)
   'set t 't
   'display z'
   if (rc!=0); break; endif;
   t = t + 1
endwhile
```

Functions

Functions may either be contained within the script file itself, or the may be intrinsic functions. Functions contained within other script files are not supported as yet (other script files may be executed via the GrADS run command).

In either case, functions are invoked as a script expression is being evaluated. Script functions always have a single string result, but may have one or more string arguments. Functions are invoked by:

name(arg,arg,arg...)

If the function has no arguments, you must still provide the parentheses:

name()

You may provide your own functions from within your script file by using the **function** definition record:

function name(variable, variable, ...)

To return from a function, use the **return** command:

return expression

The **expression** is optional; if not provided, a NULL string will be returned. (A null string is: '') The result of the function is the result of the expression specified on the return command.

When a function is invoked, the arguments are evaluated, then flow of control is transferred to the function. The variables contained in the list within the function definition record are initialized to the values of the passed arguments. If too few arguments where passed for the variables specified, the trailing variables are uninitialized. If too many arguments are passed, the extra arguments are discarded.

You may modify the variables from the function definition record without modifying the variables from the calling routine.

Scope of variables is normally local to the function, but can be global.

When a script file is first invoked (via the **run** command), execution starts at the beginning of the file. A function definition record may optionally be provided at the beginning. If it is, it should specify one variable name. This variable will be initialized to any 'run' command options. If no options were given, the variable will be initialized to NULL.

Assignment

The format of the assignment record is:

variable = expression

The expression is evaluated, and the result is assigned to be the value of the indicated variable.

Standard input/output

To write information or questions to the terminal (standard output), use the **’say’** or **‘prompt’** commands:

say expression
prompt expression

The result of the **expression** is written to the terminal. The **prompt** command works the same way as the **say** command but does not append a **carriage return**.

To read information from the terminal (standard input), use the **pull** command:

pull variable

The script pauses for user keyboard input (up to the carriage return), and the string entered by the user is assigned to the indicated variable name.

For example:

line = “Peter Pan, the flying one”
say line

To combine variables and comments when writing to standard output:

say ‘She said it is ‘ line

gives:

She said it is Peter Pan, the flying one

Sending Commands to GrADS

The statement record consists only of an expression:

expression

The expression is evaluated, and the resulting string is submitted to GrADS as a command.

After this record is executed, the script variable **'result'** is given the value of the result of the GrADS command (the result in this case is the string that GrADS would have typed to the terminal had you entered the command interactively). The script variable ’rc’ is given the return code from the GrADS command (this will always be an integer value).

The result may contain several GrADS output lines. These will be concatenated into one long string, and can be separated in the script using the **'sublin'** function.

A GrADS error resulting from an invalid command WILL NOT terminate execution of the script.

You may issue any GrADS commands from the scripting environment, including the **run** command. The result string from issuing the **run** command will be the string passed back from that ’lower level’ script via the 'return' command in that script—when that script returns to GrADS (and thus returns to the higher level script). You may recursively call any script, but you are responsible for ensuring that you can get back out of the recursion.

Intrinsic Functions

String functions

subwrd (string, word) - get a single word from a string

The result is the nth **’word’** from the **string**. If the string is too short, the result is NULL. **’word’** must be an integer.

sublin (string, line) **-** get a single line from a string containing several lines

The result is the nth **’line’** from the **string**. If the string has too few lines, the NULL string is returned. **’line’** must be an integer.

substr (string, start, length) - get part of a string

The sub-string of **string** starting at location **’start’** for length **’length’** will be returned. If the string is too short, the result will be short or NULL. **’start’** and **’length’** must be integer string values.

Input/output functions

read (filename) **-** read records from a file

The next record from file **’filename’** is read. Repeated calls may be made to read consecutive records. The result is two lines within one string. The first line is the return code, the 2nd line is the record read. The record may be a maximum of 80 characters. Use the **’sublin’** function to separate the result. Return codes are:

0 - ok
1 - open error
2 - end of file
8 - file open for write
9 - I/O error

Files are opened when the first call to read is made for a particular file name. Files are closed when the execution of the script file terminates (note that files remain open between function calls, etc).

write (filename, record <, append>) - write records to an output file

The record is written to file **’filename’**. On the first call to write for a particular file, the file is opened in write mode. This will destroy an existing file! If you use the optional append flag, the file will be opened in append mode, and all writes will be appended to the end of the file. Return codes are:

0 - ok
1 - open error
8 - file open for read

close (name)

Closes the named file. This must be done if you wish to read from a file you have been writing to. This can also be used to rewind a file. Return codes:

0 - ok
1 - file not open

Commands that complement the scripting language

There are some GrADS commands that, although not designed exclusively for scripts, are most useful when used to complement the scripting language. These include:

query <option>

Issue the **query** command with no options to see some available options. The following is a complete list of **query options**:

transform - does coordinate transformations, for example

query transform value1 value2

where **transform** is one of:

xy2w	XY coords to world coords
xy2gr	XY coords to grid coords
w2xy	world coords to XY coords
w2gr	world coords to grid coords
gr2w	grid coords to world coords
gr2xy	grid coords to XY coords
ll2xy	lat/long coords to XY coords
pp2xy	page coords to XY coords

These queries are valid ONLY AFTER something has been displayed. The transformations apply ONLY to the most recent item that has been displayed.

XY coords are inches on the page (ie, screen) where the page is 11x8.5 inches or 8.5x11 inches, depending on how GrADS was started.

World coords are lat, lon, lev, time or val, depending on what the dimension environment is when the grid was displayed. Note that time is displayed (and must be specified) in GrADS absolute date/time format. val is the value coordinate for a 1-D plot (linegraph).

Grid coordinates are the coordinates with respect to the grid being displayed. For station data sets, grid and world coordinates are equivalent except for the time dimension. Note that if you display a grid from a 'wrapped' data set, the grid numbers may be out of range of the actual file grid numbers. (A 'wrapped' data set is a data set that covers the earth in the longitude direction. Wrapping takes place automatically). The conversions are done consistently, but you may want to be sure you can handle the wrapping case if your data set is global.

Example:

You have displayed a Hovmoller diagram:

query xy2w 5.0 4.5

The response might be:

Lon = -95 Time = 00z5nov1992

define - lists currently defined variables

defval - give the value of a **defined variable** at a point.

For example, **query defval p 1 1** would give the value of the defined variable **p** at the point **1,1**. To interactively modify grid point values on a **defined** grid, **q defval** can be used in conjunction with **set defval**, for example:

define p=pr
q defval p 11

would return to the terminal (and the script variable result):

defval is 100

when **100** is the value of the define grid **p** at point 1,1.

To change the value of the define grid **p** at point **1,1**:

set defval p 1 1 65

changes the value of the define variable **p** at point **1,1** to **65**.

dims - gives the current dimension environment

file n - gives info on **file** number **n**

files - lists open files

fwrite - gives the name of the file used for **fwrite** operations

gxinfo - list graphics settings

query gxinfo is handy when trying to find the plot area, for example:

ga-> q gxinfo

might give:

Last Graphic = Line
Page Size = 11 by 8.5
X Limits = 2 to 10.5
Y Limits = 0.75 to 7.75
Xaxis = Lon Yaxis = Val
Mproj = 2

where:

Last Graphic = Line	you output a line plot
Page Size = 11 by 8.5	you're in landscape mode (the default)
X Limits = 2 to 10.5	The plot is bounded on the page between x=2 and 10.5 inches
Y Limits = 0.75 to 7.75	The plot is bounded between y=0.75 and 7.75 inches
Xaxis = Lon Yaxis = Val	What kind of axes you have
Mproj = number	Mproj is the map projection the data are displayed under.

where: **number** is

1 -scaled (no preservation of aspect ratio)

2 -latlon (2-D horiz fields, lon/lat)
3 -nps (northern polar stereo)
4 -sps (southern polar stereo)
5 -Robinson

NOTE: the Robinson projection only works when:

'set lon -180 180'
'set lat -90 90'

pos - waits for mouse click and returns **position** of mouse cursor.

shades - gives colors and levels of **shaded** contours

string xxxx - returns width of string **xxxx**.

time - gives **time range** of current open file

udft - lists the user-defined function table

set gxout findstn

This graphics output type expects three arguments via a **display** command. The first argument is a station data argument. The 2nd and 3rd arguments are the X and Y position on the screen of the desired search coordinates. GrADS will search for the nearest station to the specified X and Y position, and print the stid, lon, and lat of the station found. This should only be used when X and Y are the varying dimensions and AFTER a regular display command (that results in graphics output) is entered.

This command is primarily intended for use with a script. Note that this command is provided as an interim facility for doing this operation; a more complete facility will be provided for doing a variety of filtering and search operations. Thus, you should isolate the use of the command in your scripts in case it is necessary to change it later.

set dbuff on|off

Sets double buffer mode **on** or **off**. This allows animation to be controlled from a script. The clear command also sets double buffer mode off.

swap

Swaps buffers, when double buffer mode is **on**. If double buffer mode is **off**, this command has no effect.

The usual usage of these commands would be:

set dbuff on
loop
display something
swap
endloop
set dbuff off

Widgets

GrADS contains two types of graphical interface (GUI) widgets which may be used to implement a "point and click" interface using the scripting language.

On screen buttons

Here is a button script illustrating how to draw a button widget on the screen

```
*
*------------------------- dbutton -----------------
*
   set rgb 90 100 100 100
   set rgb 91  50 50 50
   set rgb 92 200 200 200
   function dbutton(bnum,xc,yc,dx,dy,string,oncol,offcol,facecol)
   set button 'oncol' 'facecol' 91 92 'offcol' '_bgcol' 91 92 6
   draw button 'bnum' 'xc' 'yc' 'dx' 'dy' 'string
   return
```

where:

for **set button....**

oncol	color of the text when in the "on" state (1)
facecol	color of the button face in the "on" state
offcol	color of the text when in the "off" state (0)
91 92	color of the button outline for 3-D look
_bgcol	button face color in the "off" state
6	thickness of the shadow outline

for **draw button...**

bnum	button number (1-512)
xc	x center of the button in page coordinates (inches)
yc	y center of the button in page coordinates (inches)
dx	length (x) of the button in inches
dy	height (y) of the button in inches
string	string to display in the button center at (xc,yc)

You can also redraw a button:

redraw button ### 0|1 where:

is the button number from **draw button ### ...,** and **0** or **1** is the "state"

Rubber banding

GrADS has a widget type called "**rband**" for rubber banding. There are two **modes**: 1) **box**; and 2) **line**

In **box** mode, when the user clicks and drags a box is opened and in **line** mode you get a line.

To set up the **rband**,

set rband nn mode x1 y1 x2 y2 where:

nn	- widget #
mode	= box or line
x1	- lowest point in x page units where the widget will be active
y1	- lowest point in y page units where the widget will be active
x2	- highest point in x page units where the widget will be active

y2 - highest point in y page units where the widget will be active

For example, suppose you did **q gxinfo** and you want to set up a **box** rubber band widget in the plot region only,

Last Graphic = Line
Page Size = 11 by 8.5
X Limits = 2 to 10.5
Y Limits = 0.75 to 7.75
Xaxis = Lon Yaxis = Val
Mproj = 2

You would first,

set rband 21 box 2 0.75 10.5 7.75

and then to activate the widget,

ga-> q pos

which freezes the system until the user clicks on the screen. After clicking and dragging you would get this kind of response from GrADS:

Position = 2.13125 7.565 1 2 7.08125 2.19583 where:

2.13 - x of the first corner of the box (x1)
7.56 - y of the first corner of the box (y1)
1 - which button was pressed:
- **1 -** left
- **2 -** middle
- **3 -** right

2 - widget type (rband):
- **1 -** button
- **2 -** rband

7.08 - x of the second corner of the box (x2)
7.56 - y of the second corner of the box (y2)

The page coor can be then be parsed and used in

’q xy2w’

to recover the lat/lon points...

Examples

A few simple example scripts are provided with the GrADS distribution. If you do not know where these files are, please email Brian Doty (doty@cola.iges.org) and I will send them to you. I can also send you other (longer) examples which require data sets that you won’t have, but which can provide more complex examples.

cbar.gs	Draw color bars after a shaded contour plot is displayed
xyplot.gs	Does general XY plot.
string.gs	Plots string at point-click location.
draw.gs	Draw line via point-click.

24.0 Using Map Projections in GrADS

It is important to understand the distinction between the two uses of map projections when creating GrADS displays of your data:

- projection of the data (preprojected grids);
- projection of the **display**.

GrADS supports two types of **data** grids:

- **lon/lat** grids (and not necessarily regular, e.g., **gaussian**);
- preprojected grids.

Using Preprojected Grids

Preprojected data are data **already** on a map projection. GrADS supports four types of **preprojected** data:

1. N polar stereo (NMC model projection);
2. S polar stereo (NMC model projection) ;
3. Lambert Conformal (originally for Navy NORAPS model);
4. NMC eta model (unstaggered).
5. More precise N and S polar stereo (hi res SSM/I data)
6. Colorado State University RAMS model (oblique polar stereo; beta)

When **preprojected** grids are opened in GrADS, bilinear interpolation constants are calculated and all date are displayed on an internal GrADS **lat/lon** grid defined by the **xdef** and **ydef** card in the data description or "**.ctl**" file (that's why it takes longer to "open" a **preprojected** grid data set).

It is very important to point out that the internal GrADS grid can be any grid as it is completely independent of the **preprojected** data grid. Thus, there is nothing stopping you displaying **preprojected** data on a very high res **lon/lat** grid (again, defined in the **.ctl** by **xdef** and **ydef**). In fact, you could create and open multiple **.ctl** files with different resolutions and/or regions which pointed to the same **preprojected** data file.

When you do a "**display**" (i.e., get a grid of data), the **preprojected** data are bilinearly interpolated to the GrADS internal **lat/lon** grid. For **preprojected** scalar fields (e.g., 500 mb heights), the display is adequate and the precision of the interpolation can be controlled by **xdef** and **ydef** to define a higher spatial resolution grid.

The big virtue of this approach is that all built in GrADS analytic functions (e.g., **aave**, **hcurl**...) continue to work even though the data were not originally on a **lon/lat** grid. The downside is that you are not looking directly at your data on a geographic map. However, one could always define a **.ctl** file which simply opened the data file as i,j data and displayed without the map (**set mpdraw off**). So, in my opinion, this compromise is not that limiting even if as a modeller you wanted to look at the grid—you just don't get the map background.

Preprojected vector fields are a little trickier, depending on whether the vector is defined relative to the **data** grid or relative to the **Earth**. For example, NMC polar stereo grids use winds relative to the **data** grid and thus must be **rotated** to the internal GrADS **lat/lon** grid (again defined in the **.ctl** file by the **xdef** and **ydef** cards).

The only potential problem with working with **preprojected** data (e.g., Lambert Conformal model data) is defining the projection for GrADS. This is accomplished using a **pdef** card in the data descriptor "**.ctl**" file.

Polar Stereo Preprojected Data (coarse accuracy for NMC Models)

Preprojected data on a polar stereo projection (N and S) is defined as at NMC. For the NMC NGM model GRIB data distributed via anon ftp from nic.fb4.noaa.gov, the **pdef** card is:

> **pdef isize jsize projtype ipole jpole lonref gridinc**
> **pdef 53 45 nps 27 49 -105 190.5**

(NOTE: the * in the first column of the **.ctl** file means a comment...)

where,

ipole and **jpole** are the (i,j) of the pole and **gridinc** is the dx in km.

The relevant GrADS source is:

```
void w3fb04 (float alat, float along, float xmeshl, float orient,
float *xi, float *xj) {
/*
C
C SUBPROGRAM: W3FB04         LATITUDE, LONGITUDE TO GRID COORDINATES
C   AUTHOR: MCDONELL,J.      ORG: W345       DATE: 90-06-04
C
C ABSTRACT: CONVERTS THE COORDINATES OF A LOCATION ON EARTH FROM THE
C   NATURAL COORDINATE SYSTEM OF LATITUDE/LONGITUDE TO THE GRID (I,J)
C   COORDINATE SYSTEM OVERLAID ON A POLAR STEREOGRAPHIC MAP PRO-
C   JECTION TRUE AT 60 DEGREES N OR S LATITUDE. W3FB04 IS THE REVERSE
C   OF W3FB05.
C
C PROGRAM HISTORY LOG:
C   77-05-01  J. MCDONELL
C   89-01-10  R.E.JONES   CONVERT TO MICROSOFT FORTRAN 4.1
C   90-06-04  R.E.JONES   CONVERT TO SUN FORTRAN 1.3
C   93-01-26  B. Doty     converted to C
C
C USAGE:  CALL W3FB04 (ALAT, ALONG, XMESHL, ORIENT, XI, XJ)
C
C   INPUT VARIABLES:
C     NAMES  INTERFACE DESCRIPTION OF VARIABLES AND TYPES
C     ------ --------- -----------------------------------------------
C     ALAT   ARG LIST  LATITUDE IN DEGREES (<0 IF SH)
C     ALONG  ARG LIST  WEST LONGITUDE IN DEGREES
C     XMESHL ARG LIST  MESH LENGTH OF GRID IN KM AT 60 DEG LAT(<0 IF SH)
C                     (190.5 LFM GRID, 381.0 NH PE GRID,-381.0 SH PE GRID)
C     ORIENT ARG LIST  ORIENTATION WEST LONGITUDE OF THE GRID
C                     (105.0 LFM GRID, 80.0 NH PE GRID, 260.0 SH PE GRID)
C
C   OUTPUT VARIABLES:
C     NAMES  INTERFACE DESCRIPTION OF VARIABLES AND TYPES
C     ------ --------- -----------------------------------------------
C     XI     ARG LIST  I OF THE POINT RELATIVE TO NORTH OR SOUTH POLE
C     XJ     ARG LIST  J OF THE POINT RELATIVE TO NORTH OR SOUTH POLE
C
C   SUBPROGRAMS CALLED:
C     NAMES                                                   LIBRARY
C     ------------------------------------------------------- --------
```

```
C     COS SIN                                                     SYSLIB
C
C   REMARKS: ALL PARAMETERS IN THE CALLING STATEMENT MUST BE
C     REAL. THE RANGE OF ALLOWABLE LATITUDES IS FROM A POLE TO
C     30 DEGREES INTO THE OPPOSITE HEMISPHERE.
C     THE GRID USED IN THIS SUBROUTINE HAS ITS ORIGIN (I=0,J=0)
C     AT THE POLE IN EITHER HEMISPHERE, SO IF THE USER'S GRID HAS ITS
C     ORIGIN AT A POINT OTHER THAN THE POLE, A TRANSLATION IS NEEDED
C     TO GET I AND J. THE GRIDLINES OF I=CONSTANT ARE PARALLEL TO A
C     LONGITUDE DESIGNATED BY THE USER. THE EARTH'S RADIUS IS TAKEN
C     TO BE 6371.2 KM.
C
C ATTRIBUTES:
C   LANGUAGE: SUN FORTRAN 1.4
C   MACHINE:  SUN SPARCSTATION 1+
C*/
static float radpd = 0.01745329;
static float earthr = 6371.2;

float re,xlat,wlong,r;
  re    = (earthr * 1.86603) / xmeshl;
  xlat =  alat * radpd;
  if (xmeshl>0.0) {
    wlong = (along + 180.0 - orient) * radpd;
      r     = (re * cos(xlat)) / (1.0 + sin(xlat));
      *xi    = r * sin(wlong);
      *xj    = r * cos(wlong);

  } else {
    re    = -re;
    xlat =  -xlat;
    wlong = (along - orient) * radpd;
      r     = (re * cos(xlat)) / (1.0+ sin(xlat));
      *xi   =  r * sin(wlong);
      *xj   = -r * cos(wlong);
  }
}
```

Lambert Conformal Preprojected Data

The Lambert Conformal projection (lcc) was implemented for the U.S. Navy's limited area model NORAPS. Thus, to work with your lcc data you must express your grid in the context of the Navy lcc grid. NMC has been able to do this for their AIWIPS grids and the Navy definition should be general enough for others.

A typical NORAPS Lambert-Conformal grid is described below, including the C code which sets up the internal interpolation.

The **.ctl** file is:

dset ^temp.grd
title NORAPS DATA TEST
undef 1e20
pdef 103 69 lcc 30 -88 51.5 34.5 20 40 -88 90000 90000
xdef 180 linear -180 1.0
ydef 100 linear -10 1.0
zdef 16 levels 1000 925 850 700 500 400 300 250 200 150 100 70 50 30 20 10
tdef 1 linear 00z1jan94 12hr
vars 1
t 16 0 temp
endvars

where,

103 = #pts in x
69 = #pts in y
lcc = Lambert-Conformal
30 = lat of a ref point
88 = lon of ref point (E is positive in GrADS, W is negative)
51.5 = i of ref point
34.5 = j of ref point
20 = S true lat
40 = N true lat
88 = standard lon
90000 = dx in M
90000 = dy in M

Otherwise, it is the same as other GrADS files.

Note - the **xdef/ydef** apply to the **lon/lat** grid GrADS internally interpolates to and can be anything...

The GrADS source which maps lon/lat of the GrADS internal **lon/lat** grid to i,j of the **preprojected** grid is:

```
/* Lambert Conformal conversion */
void ll2lc (float *vals, float grdlat, float grdlon,
float *grdi, float *grdj) {
/*  Subroutine to convert from lat-lon to Lambert Conformal i,j.
Provided by NRL Monterey; converted to C 6/15/94.
c          SUBROUTINE: ll2lc
c
c          PURPOSE: To compute i- and j-coordinates of a specified
c                   grid given the latitude and longitude points.
c                   All latitudes in this routine start
c                   with -90.0 at the south pole and increase
c                   northward to +90.0 at the north pole.  The
c                   longitudes start with 0.0 at the Greenwich
c                   meridian and increase to the east, so that
c                   90.0 refers to 90.0E, 180.0 is the inter-
c                   national dateline and 270.0 is 90.0W.
c
c          INPUT VARIABLES:
c
c  vals+0    reflat: latitude at reference point (iref,jref)
c
```

```
c  vals+1    reflon: longitude at reference point (iref,jref)
c
c  vals+2    iref:   i-coordinate value of reference point
c
c  vals+3    jref:   j-coordinate value of reference point
c
c  vals+4    stdlt1: standard latitude of grid
c
c  vals+5    stdlt2: second standard latitude of grid (only required
c                    if igrid = 2, lambert conformal)
c
c  vals+6    stdlon: standard longitude of grid (longitude that
c                     points to the north)
c
c  vals+7    delx:   grid spacing of grid in x-direction
c                    for igrid = 1,2,3 or 4, delx must be in meters
c                    for igrid = 5, delx must be in degrees
c
c  vals+8    dely:   grid spacing (in meters) of grid in y-direction
c                    for igrid = 1,2,3 or 4, delx must be in meters
c                    for igrid = 5, dely must be in degrees
c
c            grdlat: latitude of point (grdi,grdj)
c
c            grdlon: longitude of point (grdi,grdj)
c
c            grdi:   i-coordinate(s) that this routine will generate
c                    information for
c
c            grdj:   j-coordinate(s) that this routine will generate
c                    information for
c
*/

  float pi, pi2, pi4, d2r, r2d, radius, omega4;
  float gcon,ogcon,ahem,deg,cn1,cn2,cn3,cn4,rih,xih,yih,rrih,check;
  float alnfix,alon,x,y;
  pi = 4.0*atan(1.0);
  pi2 = pi/2.0;
  pi4 = pi/4.0;
  d2r = pi/180.0;
  r2d = 180.0/pi;
  radius = 6371229.0;
  omega4 = 4.0*pi/86400.0;

/*mf -------------- mf*/
/*case where standard lats are the same */
  if(*(vals+4) == *(vals+5)) {
    gcon = sin(*(vals+4)*d2r);
  } else {
    gcon = (log(sin((90.0-*(vals+4))*d2r))
    log(sin((90.0-*(vals+5))*d2r)))
    /(log(tan((90.0-*(vals+4))*0.5*d2r))
    log(tan((90.0-*(vals+5))*0.5*d2r)));
  }
/*mf -------------- mf*/
  ogcon = 1.0/gcon;
  ahem = fabs(*(vals+4))/(*(vals+4));
  deg = (90.0-fabs(*(vals+4)))*d2r;
  cn1 = sin(deg);
  cn2 = radius*cn1*ogcon;
```

```
  deg = deg*0.5;
  cn3 = tan(deg);
  deg = (90.0-fabs(*vals))*0.5*d2r;
  cn4 = tan(deg);
  rih = cn2*pow((cn4/cn3),gcon);
  deg = (*(vals+1)-*(vals+6))*d2r*gcon;
  xih = rih*sin(deg);
  yih = -rih*cos(deg)*ahem;
  deg = (90.0-grdlat*ahem)*0.5*d2r;
  cn4 = tan(deg);
  rrih = cn2*pow((cn4/cn3),gcon);
  check = 180.0-*(vals+6);
  alnfix = *(vals+6)+check;
  alon = grdlon+check;
  while (alon<0.0) alon = alon+360.0;
  while (alon>360.0) alon = alon-360.0;
  deg = (alon-alnfix)*gcon*d2r;
  x = rrih*sin(deg);
  y = -rrih*cos(deg)*ahem;
  *grdi = *(vals+2)+(x-xih)/(*(vals+7));
  *grdj = *(vals+3)+(y-yih)/(*(vals+8));
}
```

NMC Eta model (unstaggered grids)

The NMC eta model "native" grid is awkward to work with because the variables are on staggered (e.g., the grid for winds is not the same as the grid for mass points) *and* non rectangular (number of points in i is *not* constant with j) grids. Because any contouring of irregularly gridded data involves interpolation at some point, NMC creates "unstaggered" eta model fields for practical application programs such as GrADS. In the unstaggered grids all variables are placed on a common *and* rectangular grid (the mass points).

Wind rotation has also been added so that vector data will be properly displayed.

The pdef card for a typical eta model grid is:

pdef 181 136 eta.u -97.0 41.0 0.38888888 0.37037037

181	= #pts in x
136	= #pts in y
eta.u	= eta grid, unstaggered
-97.0	= lon of ref point (E is positive in GrADS, W is negative) [deg]
41.0	= lat of ref point [deg]
0.3888	= dlon [deg]
0.37037	**=** dlat [deg]

The source code in GrADS for the lon,lat -> i,j mapping is:

```
void ll2eg (int im, int jm, float *vals,  float grdlon, float grdlat,
      float *grdi, float *grdj, float *alpha) {

/*  Subroutine to convert from lat-lon to NMC eta i,j.

    Provided by Eric Rogers NMC; converted to C 3/29/95 by Mike Fiorino.

c           SUBROUTINE: ll2eg
c
c           PURPOSE: To compute i- and j-coordinates of a specified
c                    grid given the latitude and longitude points.
```

```
c                   All latitudes in this routine start
c                   with -90.0 at the south pole and increase
c                   northward to +90.0 at the north pole.  The
c                   longitudes start with 0.0 at the Greenwich
c                   meridian and increase to the east, so that
c                   90.0 refers to 90.0E, 180.0 is the inter-
c                   national dateline and 270.0 is 90.0W.
c
c          INPUT VARIABLES:
c
c  vals+0    tlm0d: longitude of the reference center point
c
c  vals+1    tph0d: latitude of the reference center point
c
c  vals+2    dlam:  dlon grid increment in deg
c
c  vals+3    dphi:  dlat grid increment in deg
c
c
c            grdlat: latitude of point (grdi,grdj)
c
c            grdlon: longitude of point (grdi,grdj)
c
c            grdi:   i-coordinate(s) that this routine will generate
c                    information for
c
c            grdj:   j-coordinate(s) that this routine will generate
c                    information for
c

*/

  float pi,d2r,r2d, earthr;
  float tlm0d,tph0d,dlam,dphi;
  float
   phi,lam,lame,lam0,phi0,lam0e,cosphi,sinphi,sinphi0,cosphi0,sinlamr,cos
   lamr;
  float x1,x,y,z,bigphi,biglam,cc,num,den,tlm,tph;

  int idim,jdim;

  pi=3.141592654;

  d2r=pi/180.0;
  r2d=1.0/d2r;
  earthr=6371.2;

  tlm0d=-*(vals+0); /* convert + W to + E, the grads standard for
   longitude */
  tph0d=*(vals+1);
  dlam=(*(vals+2))*0.5;
  dphi=(*(vals+3))*0.5;

  /* grid point and center of eta grid trig */

  /* convert to radians */

  phi    = grdlat*d2r;
  lam    = -grdlon*d2r; /* convert + W to + E, the grads standard for
   longitude */
  lame   = (grdlon)*d2r;

  phi0   = tph0d*d2r;
  lam0   = tlm0d*d2r;
  lam0e  = ( 360.0 + *(vals+0) )*d2r;
```

```
  /* cos and sin */

  cosphi = cos(phi);
  sinphi = sin(phi);

  sinphi0 = sin(phi0);
  cosphi0 = cos(phi0);

  sinlamr=sin(lame-lam0e);
  coslamr=cos(lame-lam0e);

  x1     = cosphi*cos(lam-lam0);
  x      = cosphi0*x1+sinphi0*sinphi;
  y      = -cosphi*sin(lam-lam0);
  z      = -sinphi0*x1+cosphi0*sinphi;

  /* params for wind rotation alpha */

  cc=cosphi*coslamr;
  num=cosphi*sinlamr;
  den=cosphi0*cc+sinphi0*sinphi;

  tlm=atan2(num,den);

  /* parms for lat/lon -> i,j */

  bigphi = atan(z/(sqrt(x*x+y*y)))*r2d;
  biglam = atan(y/x)*r2d;

  idim = im*2-1;
  jdim = jm*2-1 ;

  *grdi  = (biglam/dlam)+(idim+1)*0.5;
  *grdj  = (bigphi/dphi)+(jdim+1)*0.5;
  *grdi  = (*grdi+1)*0.5-1;
  *grdj  = (*grdj+1)*0.5-1;

  *alpha = asin( ( sinphi0*sin(tlm)) / cosphi ) ;

/*
  printf("qqq %6.2f %6.2f %6.2f %6.2f %g %g %g %g\n",
    grdlon,grdlat,*grdi,*grdj,*alpha,tlm*r2d,cosphi,sinphi0);
*/

}
```

NMC high accuracy polar stereo for SSM/I data

The polar stereo projection used by the original NMC models is not very precise because it assumes the earth is round (eccentricity = 0). While this approximation was reasonable for coarse resolution NWP models, it is inadequate to work with higher resolution data such as SSM/I.

Wind rotation has not been implemented!!! Use only for scalar fields.

pdef ni nj pse slat slon polei polej dx dy sgn

ni	= # points in x
nj	= # points in y
slat	= absolute value of the standard latitude
slon	= absolute value of the standard longitude
pse	= polar stereo, “eccentric”

polei = x index position of the pole (where (0,0) is the index of the first point vice the more typical (1,1))
polej = y index position of the pole (where (0,0) is the index of the first point vice the more typical (1,1))
dx = delta x in km
dy = delta y in km
sgn = 1 for N polar stereo and -1 for S polar stereo

Source code in GrADS for the lon,lat -> i,j mapping:

```
void ll2pse (int im, int jm, float *vals, float lon, float lat,
       float *grdi, float *grdj) {

  /* Convert from geodetic latitude and longitude to polar stereographic
     grid coordinates.  Follows mapll by V. J. Troisi.         */
  /* Conventions include that slat and lat must be absolute values */
  /* The hemispheres are controlled by the sgn parameter */
  /* Bob Grumbine 15 April 1994. */

  const rearth = 6738.273e3;
  const eccen2 = 0.006693883;
  const float pi = 3.141592654;

  float cdr, alat, along, e, e2;
  float t, x, y, rho, sl, tc, mc;
  float slat,slon,xorig,yorig,sgn,polei,polej,dx,dy;

  slat=*(vals+0);
  slon=*(vals+1);
  polei=*(vals+2);
  polej=*(vals+3);
  dx=*(vals+4)*1000;
  dy=*(vals+5)*1000;
  sgn=*(vals+6);


  xorig = -polei*dx;
  yorig = -polej*dy;

  /*printf("ppp %g %g %g %g %g %g
   %g\n",slat,slon,polei,polej,dx,dy,sgn);*/

  cdr   = 180./pi;
  alat  = lat/cdr;
  along = lon/cdr;
  e2 = eccen2;
  e  = sqrt(eccen2);

  if ( fabs(lat) > 90.)  {
    *grdi = -1;
    *grdj = -1;
    return;
  }
  else {
    t = tan(pi/4. - alat/2.) /
      pow( (1.-e*sin(alat))/(1.+e*sin(alat)) , e/2.);

    if ( fabs(90. - slat) < 1.E-3) {
      rho = 2.*rearth*t/
   pow( pow(1.+e,1.+e) * pow(1.-e,1.-e) , e/2.);
```

```
    }
    else {
      sl = slat/cdr;
      tc = tan(pi/4.-sl/2.) /
     pow( (1.-e*sin(sl))/(1.+e*sin(sl)), (e/2.) );
      mc = cos(sl)/ sqrt(1.-e2*sin(sl)*sin(sl) );
      rho = rearth * mc*t/tc;
    }

    x = rho*sgn*cos(sgn*(along+slon/cdr));
    y = rho*sgn*sin(sgn*(along+slon/cdr));

    *grdi = (x - xorig)/dx+1;
    *grdj = (y - yorig)/dy+1;

    /*printf("ppp (%g %g) (%g %g %g) %g
  %g\n",lat,lon,x,y,rho,*grdi,*grdj);*/

    return;
  }

}
```

CSU RAMS Oblique Polar Stereo Grids

The CSU RAMS model uses an oblique polar stereo projection. This projection is still being tested...

pdef 26 16 ops 40.0 -100.0 90000.0 90000.0 14.0 9.0 180000.0 180000.0

26	= #pts in x
16	= #pts in y
ops	= oblique polar stereo
40.0	= lat of ref point (14.0, 9.0)
-100.0	= lon of ref point (14.0, 9.0 (E is positive in GrADS, W is negative)
90000.0	= xref offset [m]
90000.0	= yref offset [m]
14.0	= i of ref point
9.0	= j of ref point
180000.0	= dx [m]
180000.0	= dy [m]

Wind rotation has not been implemented!!! Use only for scalar fields.

Source code in GrADS for the lon,lat -> i,j mapping:

```
void ll2ops(float *vals, float lni, float lti, float *grdi, float *grdj)
  {

  const float radius = 6371229.0 ;
  const float pi = 3.141592654;

  float stdlat, stdlon, xref, yref, xiref, yjref, delx , dely;

  float plt,pln;
  double pi180,c1,c2,c3,c4,c5,c6,arg2a,bb,plt1,alpha,
   pln1,plt90,argu1,argu2;

  double hsign,glor,rstdlon,glolim,facpla,x,y;
```

```
  stdlat = *(vals+0);
  stdlon = *(vals+1);
  xref = *(vals+2);
  yref = *(vals+3);
  xiref = *(vals+4);
  yjref = *(vals+5);
  delx = *(vals+6);
  dely = *(vals+7);

  c1=1.0 ;
  pi180 = asin(c1)/90.0;

/*
c
c     set flag for n/s hemisphere and convert longitude to <0 ; 360>
  interval
c
*/
  if(stdlat >= 0.0) {
    hsign= 1.0 ;
  } else {
    hsign=-1.0 ;
  }
/*
c
c     set flag for n/s hemisphere and convert longitude to <0 ; 360>
  interval
c
*/
  glor=lni ;
  if(glor <= 0.0) glor=360.0+glor ;
  rstdlon=stdlon;
  if(rstdlon < 0.0) rstdlon=360.0+stdlon;

/*
c
c     test for a n/s pole case
c
*/
  if(stdlat == 90.0) {
    plt=lti ;
    pln=fmod(glor+270.0,360.0) ;
    goto l2000;
  }

  if(stdlat == -90.0) {
    plt=-lti ;
    pln=fmod(glor+270.0,360.0) ;
    goto l2000;
  }


/*
c
c     test for longitude on 'greenwich or date line'
c
*/
  if(glor == rstdlon) {
    if(lti > stdlat) {
      plt=90.0-lti+stdlat;
      pln=90.0;
```

```
    } else {
      plt=90.0-stdlat+lti;
      pln=270.0;;
    }
    goto l2000;
  }

  if(fmod(glor+180.0,360.0) == rstdlon) {
    plt=stdlat-90.0+lti;
    if(plt < -90.0) {
      plt=-180.0-plt;
      pln=270.0;
    } else {
      pln= 90.0;
    }
    goto l2000 ;
  }

/*
c
c     determine longitude distance relative to rstdlon so it belongs to
c     the absolute interval 0 - 180
c
*/
  argu1 = glor-rstdlon;
  if(argu1 > 180.0) argu1 = argu1-360.0;
  if(argu1 < -180.0) argu1 = argu1+360.0;

/*
c
c     1. get the help circle bb and angle alpha (legalize arguments)
c
*/

  c2=lti*pi180 ;
  c3=argu1*pi180 ;
  arg2a = cos(c2)*cos(c3) ;
  if( -c1 > arg2a ) arg2a = -c1 ; /* arg2a = max1(arg2a,-c1)  */
  if(  c1 < arg2a ) arg2a = c1 ; /* min1(arg2a, c1)         */
  bb = acos(arg2a) ;

  c4=hsign*lti*pi180 ;
  arg2a = sin(c4)/sin(bb) ;
  if( -c1 > arg2a ) arg2a = -c1 ; /* arg2a = dmax1(arg2a,-c1) */
  if(  c1 < arg2a ) arg2a = c1  ; /* arg2a = dmin1(arg2a, c1) */
  alpha = asin(arg2a) ;
/*
c
c     2. get plt and pln (still legalizing arguments)
c
*/
  c5=stdlat*pi180 ;
  c6=hsign*stdlat*pi180 ;
  arg2a = cos(c5)*cos(bb) + sin(c6)*sin(c4) ;
  if( -c1 > arg2a ) arg2a = -c1 ; /* arg2a = dmax1(arg2a,-c1) */
  if(  c1 < arg2a ) arg2a = c1  ; /* arg2a = dmin1(arg2a, c1) */
  plt1   = asin(arg2a) ;

  arg2a = sin(bb)*cos(alpha)/cos(plt1) ;

  if( -c1 > arg2a ) arg2a = -c1 ; /* arg2a = dmax1(arg2a,-c1) */
```

```
  if(  c1 < arg2a ) arg2a =  c1 ; /* arg2a = dmin1(arg2a, c1) */
  pln1   = asin(arg2a) ;


/*
c
c    test for passage of the 90 degree longitude (duallity in pln)
c         get plt for which pln=90 when lti is the latitude
c
*/
  arg2a = sin(c4)/sin(c6) ;
  if( -c1 > arg2a ) arg2a = -c1 ; /* arg2a = dmax1(arg2a,-c1) */
  if(  c1 < arg2a ) arg2a =  c1 ; /* arg2a = dmin1(arg2a, c1) */
  plt90 = asin(arg2a) ;

/*
c
c         get help arc bb and angle alpha
c
*/
  arg2a = cos(c5)*sin(plt90) ;
  if( -c1 > arg2a ) arg2a = -c1 ; /* arg2a = dmax1(arg2a,-c1) */
  if(  c1 < arg2a ) arg2a =  c1 ; /* arg2a = dmin1(arg2a, c1) */
  bb    = acos(arg2a) ;

  arg2a = sin(c4)/sin(bb) ;
  if( -c1 > arg2a ) arg2a = -c1 ; /* arg2a = dmax1(arg2a,-c1) */
  if(  c1 < arg2a ) arg2a =  c1 ; /* arg2a = dmin1(arg2a, c1) */
  alpha = asin(arg2a) ;

/*
c
c         get glolim - it is nesc. to test for the existence of solution
c
*/
  argu2  = cos(c2)*cos(bb) / (1.-sin(c4)*sin(bb)*sin(alpha)) ;
  if( fabs(argu2) > c1 ) {
    glolim = 999.0;
  } else {
    glolim = acos(argu2)/pi180;
  }

/*
c
c     modify (if nesc.) the pln solution
c
*/
  if( ( fabs(argu1) > glolim && lti <= stdlat ) || ( lti > stdlat ) ) {
    pln1 = pi180*180.0 - pln1;
  }
/*
c
c     the solution is symmetric so the direction must be if'ed
c
*/
  if(argu1 < 0.0) {
    pln1 = -pln1;
  }
/*
c
c     convert the radians to degrees
```

```
c
*/
  plt = plt1/pi180 ;
  pln = pln1/pi180 ;


/*
c
c     to obtain a rotated value (ie so x-axis in pol.ste. points east)
c     add 270 to longitude
c
*/
  pln=fmod(pln+270.0,360.0) ;

 l2000:

/*
c
c     this program convert polar stereographic coordinates to x,y ditto
c     longitude:   0 - 360  ; positive to the east
c     latitude : -90 -  90  ; positive for northern hemisphere
c     it is assumed that the x-axis point towards the east and
c     corresponds to longitude = 0
c
c     tsp 20/06-89
c
c     constants and functions
c
*/
  facpla = radius*2.0/(1.0+sin(plt*pi180))*cos(plt*pi180);
  x = facpla*cos(pln*pi180) ;
  y = facpla*sin(pln*pi180)  ;

  *grdi=(x-xref)/delx + xiref;
  *grdj=(y-yref)/dely + yjref;

  return;

}
```

Pitfalls when using preprojected data

There are a few *gotchas* with using **preprojected** data:

1) the units in the variable definition for the **u** and **v** components **must** be **33** and **34** (the GRIB standard) respectively, e.g.,

   u 15 33 u component of the wind at 15 pressure levels
   v 15 34 v component of the wind at 15 pressure levels

2) wind rotation is handled for **polar stereo** (N and S) **preprojected** data, but *not* for Lambert Conformal, as the Navy rotates the winds relative to earth. This will have to be added later......

3) the **eta.u and ops** projection are still experimental...

GrADS Display Projections

Now that you hopefully understand GrADS **data grids**, it is time to discuss **display projections**. Graphics in GrADS are calculated relative to the internal GrADS data grid **i,j** space, **transformed** to the display device coordinates (e.g., the screen) and then displayed. That is, the **i,j** of the graphic element is converted to **lat/lon** and then to **x,y** on the screen via a map **projection**.

GrADS currently supports four **display projections**:

- lat/lon (or spherical);
- N polar stereo (**set mproj nps**);
- S polar stereo (**set mproj sps**);
- the Robinson projection (set lon -180 180, set lat -90 90, set mproj robinson).

As you can probably appreciate, the i,j-to-lon/lat-to-screen x,y for **lon/lat** displays is very simple and is considerably more complicated for N and S **polar stereo** projections.

In principle, a Lambert Conformal display projection could be implemented. It just takes work and a simple user interface for setting up that **display** projection. Actually, the user interface (i.e., "set" calls) is the most difficult problem...

Summary and Plans

GrADS handles map projections in two different ways. The first is **preprojected** data where the fields are *already* on a projection (e.g., Lambert Conformal). It is fairly straightforward to implement other **preprojected** data projections and we will be fully implementing the NMC **eta** grid both staggered and unstaggered, "thinned" **gaussian** grids and the CSU RAMS **oblique polar stereo** projection. The second is in how i,j graphics (calculated in "grid" space) are displayed on a map background. Currently, only a few basic projections (lon/lat, polar stereo and robinson) are supported, but perhaps the development group will tackle this problem.

Appendices

Appendix A: Supplementary Scripts

This Appendix includes documentation (where available) and names of available scripts to supplement GrADS utilities. Please see ftp://sprite.llnl.gov/pub/fiorino/grads/lib.

1) Correlation between two horizontal grids (corr.gs)

Author: Mike Fiorino

2) GrADS Color Table Script (cmap.gs)

Author: Mike Fiorino

Prerequisites

I assume that you know something about GrADS scripts and hopefully have written a few. In the text below strings between the ’ ’ are GrADS command and strings between " " are UNIX commands or files names.

Using Colors in GrADs

Let’s first go over using colors. GrADS includes 16 defaults colors numbered 0 15 and has the capability of extending the number of colors using

’set rgb ## R G B’ where

is the color number
R is the Red value (0-255)
G is the Green value of the color (0-255)
B is the Blue value of the color (0-255)

So, to create your own color map for colors number 21-24 you would, for example,

’set rgb 20 0 155 155’
’set rgb 21 155 0 155’
’set rgb 22 0 0 155’
’set rgb 23 155 155 155’
’set rgb 24 155 0 0’

and you could then access these colors just as you would the 0-15 built-in colors. For example,

’set gxout contour’
’set ccolor 23’
’d slp’

would contour the **slp** field using color **23**

If you wanted to set the "rainbow" sequence to your new colors,

’set rbcols 21 22 23 24’
’d slp’

would contour slp with a range of colors from 21-24

Perhaps the most useful application of user-defined colors is in color fill graphics. For example,

’set gxout shaded’
’set clevs 1000 1008 1016 1024 ’
’set ccols 0 21 22 23 24’

would shade areas < 1000 in slp in black (no colors), areas where slp is 1000 - 1008 in 21, ... , 1016 - 1024 in 23 and all values > 1024 in 24.

For grids with index or parametric data (i.e., non-continuous), the fgrid command is very useful,

’set gxout fgrid’
’set fgvals 1000 22’
’d slp’

would fill grid boxes where slp = 1000 with color 22. You can extend this by,

’set fgvals 1000 22 1008 23 1016 24’

to color more grid boxes,

An alternative to contour color fill is box color fill using,

’set gxout grfill’
’d slp’

which color fills the grid boxes using the same coloring scheme as,

’set gxout shaded’

and you can control the coloring scheme in the same way. The only problem with

’set rgb’

is that you can’t tell what the color will look like on the screen (or on hardcopy) until you test it by running GrADS. Here is where the script cmap.gs comes; it allows you to **interactively** create and **modify** a color table.

Using cmap.gs

I use the .gct file postfix to distinguish GrADS color tables from other GrADS files (e.g., .gs = GrADS scripts) and at the first invocation of cmap.gs, the a color table called "grads.gct" is created. I typically rename grads.gct to another file name (e.g., nmc.gct) and then use that table in subsequent scripts.

Let’s create a color table. First fire up GrADS in landscape mode

"grads -l"

resize your graphics window and at the GrADS terminal window type,

’run cmap.gs’

The first thing the script will say is,

"Enter the Number of Colors:

Type in a number between 1-100, for example,

’10’

You should then see on the graphics screen (--> explains what’s what):

GrADS color table for : grads.gct **-->** name of the color table file
(10 boxes) **-->** boxes with color to edit
1 2 3 4 5 6 7 8 9 10 **-->** number of the box
1 **-->** color number being edited

```
 ---       ---         ---
  |         |           |
  |         |           |
  |         |           |         -->  sliders to control R G B
  |         |           |
  |         |           |
  |         |           |
  |0        |0          |0        -->  the value of R G B
 ---        ---        ---


|--------|
| Save & |         -->  box to click to save and quit
|  Quit  |
|--------|
```

To edit color #2, use your mouse and click on the box above number 2. The color number will change to 2 and you're ready to edit. Click just to the left of the slider to change the value. The bottom part of the slider is 0 and the top is 255. Just play with each slider until you are happy with the color and go on to another color. Repeat the "click to the left of the slider process" and when you are all done, click on the save and quit button. This will save your color table to the file "grads.gct". Here is what it will look something like:

```
1 225 174 91
2 238 214 129
3 163 233 0
4 0 0 88
5 0 201 0
6 0 213 107
7 240 192 69
8 233 144 227
9 221 192 109
10 247 0 0
```

To access these colors in GrADS use the **colortab** function provided at the end of doc. Here's how the "grads.gct" file color table is accessed in a GrADS script:

'rc=colortab(grads)'

rc is a return code and equals the number of colors in the color table file "grads.gct", if the file was there and was readable. Note that the ".gct" in the file name is implicit. Even though the numbers are referenced 1-10 in cmap.gs, in GrADS the colors are numbered as 21-30. The starting number of the color table is arbitrary; I chose 21 to separate user-defined colors from the 0-15 GrADS default colors *which cannot be changed.*

To use the colors try something like,

'set gxout shaded'
'set clevs 1000 1004 1008 1016'

’set ccols 0   21   22   23   24’

Undoubtedly you will not be happy with your first color table. To edit it, just rerun cmap.gs, but use the file name as a command line parameter to cmap.gs, e.g.,

"grads"
’run cmap.gs. grads’

The color table will be read in and you can now edit the colors and save it. However, cmap.gs will overwrite the file, so copy it to another file if you want to keep the original.

Problems and questions

The colors do not come out right on a PC running Xvision during editing, but not when running GrADS. The problem is in the X server because when you have draw color 20 on the screen and then, **’set rgb 20’** to a different color, *it changes on the screen* (the difference between pseudo and true color in X).

3) Font Display (font.gs)

Displays a font set in GrADS. All characters for the font corresponding to the calling argument are shown (default to font set 1). For example:

run font.gs 2   displays font set 2.

4) Plot a color bar (cbar.gs)

Plots a color bar key next to the map when **gxout = shaded**.

5) Stack commands and display on flush (stack.gs)

Useful for PC GrADS. Delays display until a sequence of commands has been entered and then executes them sequentially on flush.

6) Draw all WX Symbols (wxsym.gs)

Displays available weather symbols.

7) (draw.gs)

Author: Mike Fiorino

8) (string.gs)

Author: Mike Fiorino

9) (loop.gs)

Author: Mike Fiorino

10)   (bsamp.gs)

Author: Brian Doty

11) Expanded Color Bar Script (cbarn.gs)

Author: Mike Fiorino

Here is my version of the **cbar.gs** script. I call it **cbarn.gs** and is considerably more powerful that the original. It allows you to scale and place the colorbar anywhere on the page and is visually more appealing.

12) Computing Standard Deviation (sd.gs)

Author: Anjuli S Bamzai

This is an example script outlining the procedure to be used to write a script generating standard deviations.

I recently wrote a fairly long script that took a time series of weekly global gridded data and calculated seasonal std dev., monthly data should be along the same lines... calculate climo and time series of anomalies from the series first.

Let 's assume you want std dev of slp for Jan from your series. Here I use the notation **slpanom** for the time series of the slp anomalies you need to generate first. **sd.gs** would give us the sum of the squares of the anomalies after the **do loop**.   **nJan** is the number of Januarys that occurred in the entire time series. **sdJan** for the std dev of slp for Jan..is your final result

The basic trick is to set a do loop that goes over the entire series and use commands such as in **sd.gs**,

13) Draw an x,y Plot (xyplot.gs)

Author: Mike Fiorino

Appendix B: Using GRIB Data in GrADS

Gribscan

The "**gribscan**" routine is used for extracting grid info from GRIB data files and features:

- grid output in ASCII, floats, and/or grib;
- product/grid information;
- automatic "scanning" for GRIB records so that you don’t have to know the physical layout of the data to scan it.

File options:

i ifname = input grib file name
o ofname = output file name WITHOUT an extension
og = output GRIB
oa = output ASCII (%8g in C)
of = a stream of floats. This is machine dependent and = 64-bit on Crays & 32-bit elsewhere

If **-i** is not invoked then gribscan asks for a file name. If **-o** is omitted then a default **ofname** of **zy0x1w2.type** is created where **type** =

asc - ascii
grb - GRIB
dat - a stream of floats (GrADS format)

The FNMOC folks will "get" the **zy0x1w2** name...

Processing Options:

hNNN

fixed file header of **NNN** bytes. The default is to **seek** the first GRIB message automatically, but if you know **NNN**, it is more efficient to specify it.

sNNN

max number of bytes between GRIB messages in the file, the default is 500 and it is assumed that you want to ignore junk (e.g., comm stuff) between data.

spNNN = select parameter # **NNN** (e.g., **-sp11** for temperature)
slNNN = select level # **NNN** (e.g., **-sp500** to get 500 mb fields)
stNNN = select tau # **NNN** (e.g., **-st12** to get t=12 forecasts)

The **-s?** options can be strung together to output a very narrow set of fields. For example to only output the 500 mb u component at t=48 use:

sp33 -sl500 -st48

Special note to NMC users

The once "standard" 81-byte header in an NMC GRIB file contained the string **GRIB**.

Unfortunately, the same string is part of the GRIB indicator section itself! Thus, an automatic scan for GRIB to demark the start of the data will fail if the 81-byte header is present!

Thus, you have to know that avn flux files have the 81 byte header and run it as,

gribscan -h81

When in doubt (or failure) try **-h81 -v**.

Display options:

q = quick output to extract stuff GrADS gribmap cares about
q1 = one-line quick output
d = common delimited mode
v = verbose mode for diagnostics
bd = binary data section info
gv = use the NMC GRIB variable table to output the mnemonic, title and units from the standard NMC table
gd = output info from the grid defn sec
S = Silent NO standard output

Some examples:

First,

cd /cray3_com_eta/PROD/erl.940829

Then,

1) A "quick" scan to get the info GrADS cares about:

gribscan -q -i eta.T12Z.PGrbF48 | grep 184 :
184, F ,135,108,100,0,100,0,1e+09, T ,1994,8,29,12,0,1,48,0, G ,104, BDTG, 94082912

184	**-**	field # in the file
F	**-**	field data
135	**-**	param #
108	**-**	level indicator
100	**-**	level
0	**-**	l1 byte 1 of level
100	**-**	l2 byte 2 of level
0	**-**	time range indicator
1e+09	**-**	decimal scale factor
T	**-**	time data follows
1994	**-**	year
8	**-**	month
29 -	day	
12 -	hour	
0	**-**	min
1	**-**	forecast time unit (hour)
48 -	t=48 h forecast	

G	**-**	grid param follows
104	**-**	NMC grid #104
BDTG	**-**	Base date-time-group (yymmddhh) follows

2) Comma delimited output for parsing by things like awk:

gribscan -d -i eta.T12Z.PGrbF48 | grep 184 :
PDS,184,104,135,108,100,0,100,1994,8,29,12,0,1,48,0,0,1e+09

same as above but arranged differently

3) A full listing:

gribscan -d -gv -bd -gd -i eta.T12Z.PGrbF48 | grep 184 :
PDS,184,104,135,108,100,0,100,1994,8,29,12,0,1,48,0,0,1e+09,mconv,Horizontal moisture divergence,[kg/kg/s],GDS,5,147,110,-139.475,90.755,0.354 ,-0.268,-105.000,33536.000,0, 1,0,BDS,12, -646.844,16170,4825059,26366 where

104	- grid id
param #135	**-** mconv,Horizontal moisture divergence,[kg/kg/s] (shown by **-gv** option)
BDS	- binary data section
646.844	- ref value
16170	- # of points
4825059	- starting byte of the data
26366	- length of the grib message

N.B. not using the **-d** gives a fixed-column type output...

4) Output a selected few fields in GRIB:

gribscan -og -sp135 -q -i eta.T12Z.PGrbF48 -o /wd2/wd20/wd20mf/tmp/eta.135

Writes out all GRIB message containing the 135 parameter to the file **/wd2/wd20/wd20mf/tmp/eta.135.grb**. A subsequent gribscan on **eta.135.grb** :

gribscan -q -i eta.135.grb :
1, F ,135,108,100,0,100,0,1e+09, T ,1994,8,29,12,0,1,48,0, G ,104, BDTG, 94082912
2, F ,135,108,21860,85,100,0,1e+09, T ,1994,8,29,12,0,1,48,0, G ,104, BDTG, 94082912

Gribmap

When you set up a GrADS data descriptor file (e.g., the ".ctl" file), you are defining, external to the data itself, a structure, -- how many variables, how times in a file (or set of files with the template option), the spatial dimension or "shape" of the variables, etc. The "GrADS" format (floats, either 64-bit or 32-bit IEEE depending on platform) is so simple that the relationship between the data structure defined in the .ctl file is calculated and stored in memory when the file is opened.

What makes GRIB so painful is that there is NO relationship between the GRIB data and the bigger structural context implied by the .ctl file. Hence, the need for a utility which "maps" between the GRIB data and the GrADS data description.

How this actually happens in gribmap is that each field in the GRIB data file is read and its parameters (variable, level, time, etc.) are extracted and compared to ALL the variables at any of the levels/times/UNITS in the .ctl file until a match (hopefully) is found.

The new features of gribmap allow restrictions to be placed on this matching process.

However, the first improvement in version 1.5.1 is that it supports both GRIB0 and GRIB1.... (version 0 and version 1).

Second the code now automatically scans for character string "GRIB" vice having to worry about headers and what not (e.g., "junk" between the beginning and end of the GRIB message). That is unless you are NMC and put (duh) GRIB in the header. The default scan limit is 1000 which can be changed via the command line option:

sxxxxx where **xxxxx** is the max number of bytes to search between records for GRIB.

To bypass the bytes before starting the scan process:

hxxx where **xxx** is the number of bytes, or for nmc:
hnmc

Other features invoked at the command line include:

- **v** nicer output to verify what you are attempting to map...
- **t0** a match can only occur if the base time in the grib record is the same as the initial time in the .ctl file. This is used to pull out a forecast sequence (0, 12, 24, ... ,72 h) starting a specific time (e.g., 95010300)
- **fxxx** where **xxx** is the forecast time in hours. In this case, a match occurs **only** if the **forecast** time in the grib record matches **xxx** (hours). This is used to isolate a sequence of forecasts, e.g., all the 120 h forecasts verifying during the period 00z1jan1995 to 12Z2jan1995 from the MRF ensemble runs.
- **0** ignore the forecast time in setting up the match... This is useful in reanalysis where some of the diagnostic fields are "valid" at slightly different forecast time even though the share the same starting time.

Here's a nice trick. To verify what is mapped during the gribmap:

gribmap -v -t0 ..... | grep MATCH all records matching will be displayed...

Another feature was added to map by the GRIB "time-range-indicator" as specified in the .ctl file. This was put in for handling NMC reanalysis data where the time-range-indicator distinguishes between monthly mean variances and means.

Here's an example from reanalysis (**flux.ctl**):

```
dset /d2/reanal/nmc/output/grib.v02/month.flux.%y2%m2.grb
undef 1.0e20
dtype grib
index ^flux.gmp
title NMC-NCAR reanalysis flux/gaussian grid quantities
options yrev template
xdef 192 linear 0 1.875
ydef  94 levels -88.54195 -86.65317 -84.75323 -82.85077 -80.94736
79.04349    -77.13935    -75.23505    -73.33066    -71.42619
69.52167    -67.61710    -65.71251    -63.80790    -61.90326
59.99861    -58.09395    -56.18928    -54.28460    -52.37991
50.47522    -48.57052    -46.66582    -44.76111    -42.85640
40.95169    -39.04697    -37.14225    -35.23753    -33.33281
31.42809    -29.52336    -27.61863    -25.71391    -23.80917
21.90444    -19.99971    -18.09498    -16.19025    -14.28551
12.38078    -10.47604    -8.571312    -6.666580    -4.761841
```

```
2.857109     -0.9523697     0.9523621      2.857101      4.761833
6.666565       8.571304       10.47604       12.38077       14.28551
16.19024       18.09497       19.99970       21.90443       23.80917
25.71389       27.61862       29.52335       31.42808       33.33280
35.23752       37.14224       39.04697       40.95168       42.85638
44.76111       46.66580       48.57051       50.47520       52.37990
54.28459       56.18927       58.09395       59.99860       61.90326
63.80789       65.71249       67.61710       69.52165       71.42618
73.33064       75.23505       77.13934       79.04347       80.94736
82.85077       84.75322       86.65315       88.54195
zdef  1 linear 1 1
tdef 84 linear jan1985 1mo
vars 54
    ps       0   1,  1,  0,113  Pressure [Pa]
    tg       0  11,  1,  0,113  Ground Temperature [K]
    tas      0  11,105,  2,113  2m Temperature [K]
    tg300    0  11,111,300,113  Ground Temperature 300 cm down [K]
    tg10     0  11,112, 10,113  Ground Temperature 10 cm down[K]
    tg200    0  11,112,2760,113  Ground Temperature 10-200 cm down [K]
    tcll      0  11,213,  0,113  Cloud Temperature Low [K]
    tclm       0  11,223,  0,113  Cloud Temperature Mid [K]
    tclh       0  11,233,  0,113 Cloud Temperature High [K]
    tasmax     0  15,105,  2,113  Maximum temperature [K]
    tasmin     0  16,105,  2,113  Minimum temperature [K]
    uas      0  33,105, 10,113  10m u wind [m/s]
    vas      0  34,105, 10,113  10m v wind [m/s]
    huss     0  51,105,  2,113  2m Specific humidity [kg/kg]
    pr       0  59,  1,  0,113  Precipitation rate [kg/m**2/s]
    snm      0  65,  1,  0,113  Water equiv. of accum. snow depth [kg/m**2]
    clt      0  71,200,  0,113  Total cloud cover [percent]
    cll      0  71,214,  0,113  Total cloud cover [percent]
    clm      0  71,224,  0,113  Total cloud cover [percent]
    clh      0  71,234,  0,113  Total cloud cover [percent]
    albds   0  84,  1,  0,113  Albedo [percent]
    mrro     0  90,  1,  0,113  Runoff [kg/m**2]
    sic      0  91,  1,  0,113  Ice concentration (ice=1; no ice=0) [1/0]
    rss      0 111,  1,  0,113  Net short wave radiation (surface) [W/m**2]
    rls      0 112,  1,  0,113  Net long wave radiation (surface) [W/m**2]
    hfls     0 121,  1,  0,113  Latent heat flux [W/m**2]
    hfss     0 122,  1,  0,113  Sensible heat flux [W/m**2]
    tauu     0 124,  1,  0,113  Zonal component of momentum flux [N/m**2]
    tauv     0 125,  1,  0,113  Meridional component of momentum flux [N/m**2]
    mrso10   0 144,112, 10,113  Volumetric soil moisture content 10 cm down[fraction]
    mrso200  0 144,112,2760,113  Volumetric soil moisture content 10-200cm down [fraction]
    pevpr    0 145,  1,  0,113  Potential evaporation rate [w/m**/]
    gwdu     0 147,  1,  0,113  Zonal gravity wave stress [N/m**2]
    gwdv     0 148,  1,  0,113  Meridional gravity wave stress [N/m**2]
    gflux    0 155,  1,  0,113  Ground heat flux [W/m**2]
    rsuscs   0 160,  1,  0,113  Clear sky upward solar flux [W/m**2]
    rsutcs   0 160,  8,  0,113  Clear sky upward solar flux [W/m**2]
```

```
    rsdtcs   0 161,  1,  0,113  Clear sky downward solar flux [W/m**2]
    rlutcs   0 162,  8,  0,113  Clear sky upward long wave flux [W/m**2]
    rldscs   0 163,  1,  0,113  Clear sky downward long wave flux [W/m**2]
    crfss    0 164,  1,  0,113  Cloud forcing net solar flux at sfc [W/m**2]
    crfsa    0 164,200,  0,113  Cloud forcing net solar flux in atmos [W/m**2]
    crfst    0 164,  8,  0,113  Cloud forcing net solar flux at top [W/m**2]
    crfls    0 165,  1,  0,113  Cloud forcing net long wave flux at sfc [W/m**2]
    crfla    0 165,200,  0,113  Cloud forcing net long wave flux in atmos [W/m**2]
    crflt    0 165,  8,  0,113  Cloud forcing net long wave flux at top [W/m**2]
    rsds     0 204,  1,  0,113  Downward solar radiation flux at sfc [W/m**2]
    rsdt     0 204,  8,  0,113  Downward solar radiation flux at top [W/m**2]
    rlds     0 205,  1,  0,113  Downward long wave radiation flux at sfc[W/m**2]
    rsus     0 211,  1,  0,113  Upward solar radiation flux at sfc [W/m**2]
    rsut     0 211,  8,  0,113  Upward solar radiation flux at top [W/m**2]
    rlus     0 212,  1,  0,113  Upward long wave radiation flux at sfc [W/m**2]
    rlut     0 212,  8,  0,113  Upward long wave radiation flux at top [W/m**2]
    prc      0 214,  1,  0,113  Convective precipitation rate [kg/m**2/s]
  endvars
```

The fourth units parameter in the variable description is 113 for a mean, for variances:

```
    .uas     0  33,105, 10,118  10m u wind [m/s]
    vas      0  34,105, 10,118  10m v wind [m/s]
    huss     0  51,105,  2,118  2m Specific humidity [kg/kg]
  endvars
```

If you don’t understand the time range indicator, then consult the GRIB Gospel according to John Stackpole (that is; NMC Office Note 388).

Appendix C: Command line editing and history under UNIX

If the **readline** library compiles on your system then the default prompt will be **ga->** as opposed to **ga>.** This indicates that command line editing is active...

So far, the GNU free library "**readline**" has only been compiled on the Sun and recently under on the SGIs, so you may not have it on your system.

The library defaults to **emacs** mode but can be set up to run using **vi** syntax:

Here’s a list of the commands which may typically be used:

ctrl-a	go to beginning of line
ctrl-e	go to end of line
ctrl-f	go forward one char
ctrl-b	go backward one char
ctrl-d	delete the char
ctrl-p	recall previous line
ctrl-n	recall next line
ctrl-r	reverse search

You also get **file name completion** using the **tab** key. If there is more than one option, then **double tab** will list the available completions.

For example, suppose you are running grads on div40-2 at FNMOC and want to start looking for files to open...

Type "**open/ h**" and get,

```
ga-> open /h
```

and hit two **tabs** and get:

```
h     home   home1  home2
```

then type "**ome1**" and **tab tab** and get,

```
ga-> open /home1/
GCC       bogus603    gnu       iqpops      nmcobs     roesserd
GRIB      cstrey      grads     lost+found  pacek      tsai
Mosaic    dh          hamilton  mendhall    picardr    witt
NEWDBS    dolan       hout      nicholso    qcops
```

then type "**GR**", **tab** to go to GRIB dir, followed by "**d**", **tab** to go to the dat dir and then "**n**", **tab tab** gives,

```
ga-> open /home1/GRIB/dat/nogaps.25.
nogaps.25.95021600.grb       nogaps.25.95021912.grb
nogaps.25.95021600.gribmap   nogaps.25.95021912.gribmap
nogaps.25.95021612.anal.grb  nogaps.25.anal.ctl
nogaps.25.95021612.ctl       nogaps.25.anal.gribmap
nogaps.25.95021612.grb       nogaps.25.ls.mask.ctl
nogaps.25.95021612.gribmap   nogaps.25.ls.mask.dat
nogaps.25.95021700.anal.grb
nogaps.25.95021700.ctl
```

and type "**950217**" to get

```
ga-> open /home1/GRIB/dat/nogaps.25.950217
nogaps.25.95021700.anal.grb  nogaps.25.95021712.ctl
nogaps.25.95021700.ctl       nogaps.25.95021712.grb
nogaps.25.95021700.grb       nogaps.25.95021712.gribmap
nogaps.25.95021700.gribmap
nogaps.25.95021712.anal.grb
```

and finally open the **12Z** data with **12.c, tab, return** to open the file

nogaps.25.95021712.ctl

WARNING

There is no guarantee that these **readline** routines will always work, so the **-h** option has been added to the invocation of GrADS to turn them off...

Thus,

grads -h

will give you the standard prompt,

ga>

as opposed to the command line editing prompt of

ga->

Appendix D: 32-bit IEEE floats on a Cray

This facility allows 32-bit IEEE float data, created on the cray or a BIG ENDIAN workstation (e.g., sun or sgi), to be read on the Cray. Thus, one can create binary data on the cray and work with it on *any other* platform (you have to add "big_endian" on the data descriptor file options card to work with the data on a little endian platform such as the PC). Implemented for NMC and FNMOC.

On the Cray use the following in the options card

option cray_32bit_ieee, e.g., to work with a float data from a Sun on the Cray:

change this **.ctl** file:

```
dset ^zg500.nmc_rnl.ll.sc.av.54.dat
title NMC (V02)reanalysis (T62L28 / SSI)
undef 1e20
xdef 72 linear 0 5
ydef 46 linear -90 4
zdef 1 levels 500
tdef 4 linear jan79 3mo
vars 1
   zg 0 0  height [m]
endvars
```

to:

```
dset ^zg500.nmc_rnl.ll.sc.av.54.dat
title NMC (V02)reanalysis (T62L28 / SSI)
options cray_32bit_ieee
undef 1e20
xdef 72 linear 0 5
ydef 46 linear -90 4
zdef 1 levels 500
tdef 4 linear jan79 3mo
vars 1
   zg 0 0  height [m]
endvars
```

The Cray_32bit_iee *has no effect* on a 32-bit platform. Further, if file is *not* opened if there is an invalid parameter in the options card.

Appendix E: Using GrADS on the IBM PC

Hardware considerations

GrADS for the PC operates in 32-bit mode and is compiled with the WATCOM C compiler, and comes with the DOS extender DOS4GW.EXE. This version requires a 386 or 486 machine with a math coprocessor (note that the 486DX chip includes a math coprocessor).

The supplied **'grads.exe'** and **'gxtran.exe'** will operate at various screen resolutions, depending on the setting of the environment variable GAVIDEO:

set gavideo=vga	VGA, 16 color, 640x480 (the default)
set gavideo=ega	EGA, 16 color, 640x350
set gavideo=vga256	VGA, 256 color, 640x480
set gavideo=svga	SVGA, 256 color, 800x600
set gavideo=xvga	SVGA, 256 color, 1024x768

You should not specify resolutions above VGA resolution and colors above 16 colors unless your graphics card and monitor are capable of handling such. One compiler vendor warns that it may be possible to damage your monitor if it is not capable of the requested resolution.

Some limitations of the PC version:

- does not support mouse point-and-click. The 'q pos' command returns -999 -999.
- does not support animation
- the text and graphics windows are 'shared', thus the text can overlay the graphics. This can be overcome to some extent by using one of the scripts:

  stack.gs - allows you to enter commands until you enter the **'flush'** command, then displays the results of those commands without overlaid text.

  frame.gs - 'separates' the text and graphics windows by clearing the screen when a **display** command is entered, issuing the **display** command, then waiting for you to press **enter** before clearing the screen and returning to text mode.

  Note that even though the text appears on the screen with the graphics, it will not appear on a hardcopy plot were the frame printed.

Data sets from other platforms

Binary gridded data sets may be moved from any UNIX machine to the PC and displayed using GrADS. The PC has a different byte order than most UNIX environments, such as Sun, IBM, Iris, and CRAY IEEE. The PC has the same byte order as the DECstations. Simply move the GrADS format gridded data set in binary mode, then if needed put the **OPTIONS BYTESWAPPED** keyword in the data descriptor file. See **Chapter 4** for more info.

Printing on non-postscript printers

If you do not have a postscript printer available, you may want to obtain **ghostscript**, a utility for printing postscript files on non-postscript printers. This utility is available from various anonymous ftp sources and works extremely well.

Incorporating GrADS pictures into PC software

Ghostscript comes with a utility, **ps2epsi**, which will convert postscript files into encapsulated postscript (.eps) suitable for importing into graphics software. Unfortunately, many of the graphics import filters expect Adobe Illustrator compatible .eps files and the import fails. There are two solutions to this problem. You can convert your picture to a bitmap using Ghostscript or any screen-capture utility. If you need to edit the picture, or re-size it, you need a program like CorelDraw 5 which imports postscript files directly (you do not need to use the Ghostscript utility). This works well for pictures without shading, for example, contour plots, and for shaded output to a colour printer. Shaded pictures output to a Laserjet using greyscales, however, have not the same quality as when output to a postscript printer.

The beta version 1.5x of GrADS comes with a very useful MS Windows utility, **gv.exe**. This will import GrADS metafiles directly and these can then be pasted into other Windows applications. Although at a fairly early stage of development, **gv** offers the best solution so far to printing and editing your GrADS pictures on a monochrome laser printer.

Appendix F: GrADS-related network facilities

There are a number of network sites offering various support facilities for GrADS users. These are listed in no particular order of importance, but a subscription to the listserver is highly recommended as a first step towards obtaining up-to-the-minute information, as well as help with particular problems/techniques. Omissions are unintentional, these are the ones I know about! Please let me know of any other useful sites by email (t.holt@uea.ac.uk).

Tom Holt, September 1995.

ftp Sites

The following ftp sites contain upgrades, documentation etc. on the various versions of GrADS for all available platforms.

grads.iges.org

This is the GrADS ftp site containing official documentation and executables from Brian Doty, the author of GrADS. On connection, you are immediately in the main GrADS directory.

sprite.llnl.gov

This site contains documentation and executables for development versions of GrADS produced by the GrADS development team and Mike Fiorino. These are likely to change before being incorporated into the latest “official” version of GrADS. Even so, there is a lot of very useful information here and the versions of GrADS are very usable. This documentation is based on a version of GrADS obtained from this site.

On connection, do **cd pub/fiorino/grads** to get to the GrADS stuff.

Listserver

listserv@icineca.cineca.it

The listserver automatically despatches messages to all subscribed users. Typically messages contain a request for help and, hopefully, solutions to the request from users who have solved the problem. Reference is frequently made to upgrades to GrADS and how to obtain them.

Using email, send a message to the above address containing just the words **help subscribe** for details on how to register with the listserver.

Once subscribed, please read the important information regarding sending messages etc. very carefully and store it in a safe place on your computer.

WWW Sites

There are a number of World Wide Web sites offering GrADS information.

The following two sites are on the COLA (Center for Ocean-Land-Atmosphere Studies) web server.

http://grads.iges.org/grads/head.html

Is the GrADS home page.

http://grads.iges.org/home.html

Is the COLA home page, containing GrADS-generated weather and climate maps. COLA are also attempting to provide links to other GrADS-related servers.

http://dao.gsfc.nasa.gov/grads_listserv/INDEX.html

Maintains an archive of postings to the GrADS listserver.